libdwarf

Generated by Doxygen 1.9.1

1 A Consumer Library Interface to DWARF 1
1.1 Suggestions for improvement are welcome. 1
1.2 Introduction 2
1.3 Thread Safety 2
1.4 Error Handling in libdwarf 2
1.4.1 Error Handling at Initialization 3
1.4.2 Error Handling Everywhere 4
1.4.2.1 DW_DLV_OK 4
1.4.2.2 DW_DLV_NO_ENTRY 4
1.4.2.3 DW_DLV_ERROR 4
1.4.2.4 Slight Performance Enhancement 4
1.5 Extracting Data Per Compilation Unit 5
1.6 Line Table Registers 5
1.7 Reading Special Sections Independently 5
1.8 Special Frame Registers 6
1.9 .debug_pubnames etc DWARF2-DWARF4 7
1.10 Reading DWARF with no object file present 7
1.11 Section Groups. Debug Fission. COMDAT groups 9
1.12 Details on separate DWARF object access 10
1.13 Linking against libdwarf.a 11
1.14 Suppressing CRC calculation for debuglink 12
1.15 Recent Changes 12

2 JIT and special case DWARF 17
2.1 Reading DWARF not in an object file 17
2.1.1 Describing the Interface 19
2.1.2 Describing A Section 19
2.1.3 Function Pointers 20

3 dwarf.h 23

4 libdwarf.h 25

5 checkexamples.c 27

6 Module Index 29
6.1 Modules 29

7 Data Structure Index 31
7.1 Data Structures 31

8 File Index 33
8.1 File List 33

9 Module Documentation 35

9.1 Basic Library Datatypes Group . . . 35
9.1.1 Detailed Description . . . 35
9.1.2 Typedef Documentation . . . 35
9.1.2.1 Dwarf_Unsigned . . . 35
9.1.2.2 Dwarf_Signed . . . 35
9.1.2.3 Dwarf_Off . . . 36
9.1.2.4 Dwarf_Addr . . . 36
9.1.2.5 Dwarf_Bool . . . 36
9.1.2.6 Dwarf_Half . . . 36
9.1.2.7 Dwarf_Small . . . 36
9.1.2.8 Dwarf_Ptr . . . 36
9.2 Enumerators with various purposes . . . 37
9.2.1 Detailed Description . . . 37
9.2.2 Enumeration Type Documentation . . . 37
9.2.2.1 Dwarf_Ranges_Entry_Type . . . 37
9.2.2.2 Dwarf_Form_Class . . . 37
9.3 Defined and Opaque Structs . . . 38
9.3.1 Detailed Description . . . 39
9.3.2 Typedef Documentation . . . 39
9.3.2.1 Dwarf_Form_Data16 . . . 39
9.3.2.2 Dwarf_Sig8 . . . 39
9.3.2.3 Dwarf_Block . . . 39
9.3.2.4 Dwarf_Locdesc_c . . . 40
9.3.2.5 Dwarf_Loc_Head_c . . . 40
9.3.2.6 Dwarf_Dsc_Head . . . 40
9.3.2.7 Dwarf_Frame_Instr_Head . . . 40
9.3.2.8 dwarf_printf_callback_function_type . . . 40
9.3.2.9 Dwarf_Cmdline_Options . . . 40
9.3.2.10 Dwarf_Str_Offsets_Table . . . 40
9.3.2.11 Dwarf_Ranges . . . 41
9.3.2.12 Dwarf_Regtable_Entry3 . . . 41
9.3.2.13 Dwarf_Regtable3 . . . 42
9.3.2.14 Dwarf_Error . . . 42
9.3.2.15 Dwarf_Debug . . . 42
9.3.2.16 Dwarf_Die . . . 42
9.3.2.17 Dwarf_Debug_Addr_Table . . . 43
9.3.2.18 Dwarf_Line . . . 43
9.3.2.19 Dwarf_Global . . . 43
9.3.2.20 Dwarf_Type . . . 43
9.3.2.21 Dwarf_Attribute . . . 43
9.3.2.22 Dwarf_Abbrev . . . 43
9.3.2.23 Dwarf_Fde . . . 43

9.3.2.24 Dwarf_Cie . . . 44
9.3.2.25 Dwarf_Arange . . . 44
9.3.2.26 Dwarf_Gdbindex . . . 44
9.3.2.27 Dwarf_Xu_Index_Header . . . 44
9.3.2.28 Dwarf_Line_Context . . . 44
9.3.2.29 Dwarf_Macro_Context . . . 44
9.3.2.30 Dwarf_Dnames_Head . . . 44
9.3.2.31 Dwarf_Handler . . . 44
9.3.2.32 Dwarf_Obj_Access_Interface_a . . . 45
9.3.2.33 Dwarf_Obj_Access_Methods_a . . . 45
9.3.2.34 Dwarf_Obj_Access_Section_a . . . 45
9.3.2.35 Dwarf_Rnglists_Head . . . 45
9.4 Default stack frame #defines . . . 45
9.4.1 Detailed Description . . . 46
9.5 DW_DLA alloc/dealloc typename&number . . . 46
9.5.1 Detailed Description . . . 46
9.6 DW_DLE Dwarf_Error numbers . . . 47
9.6.1 Detailed Description . . . 56
9.6.2 Macro Definition Documentation . . . 56
9.6.2.1 DW_DLE_LAST . . . 56
9.7 Libdwarf Initialization Functions . . . 56
9.7.1 Detailed Description . . . 57
9.7.2 Initialization And Finish Operations . . . 57
9.7.3 Function Documentation . . . 57
9.7.3.1 dwarf_init_path() . . . 57
9.7.3.2 dwarf_init_path_a() . . . 58
9.7.3.3 dwarf_init_path_dl() . . . 59
9.7.3.4 dwarf_init_path_dl_a() . . . 60
9.7.3.5 dwarf_init_b() . . . 60
9.7.3.6 dwarf_finish() . . . 61
9.7.3.7 dwarf_object_init_b() . . . 61
9.7.3.8 dwarf_object_finish() . . . 62
9.7.3.9 dwarf_set_tied_dbg() . . . 62
9.7.3.10 dwarf_get_tied_dbg() . . . 63
9.8 Compilation Unit (CU) Access . . . 63
9.8.1 Detailed Description . . . 64
9.8.2 Function Documentation . . . 64
9.8.2.1 dwarf_next_cu_header_e() . . . 64
9.8.2.2 dwarf_next_cu_header_d() . . . 65
9.8.2.3 dwarf_siblingof_c() . . . 66
9.8.2.4 dwarf_siblingof_b() . . . 67
9.8.2.5 dwarf_cu_header_basics() . . . 67

9.8.2.6 dwarf_child() . . . 68
9.8.2.7 dwarf_dealloc_die() . . . 69
9.8.2.8 dwarf_die_from_hash_signature() . . . 69
9.8.2.9 dwarf_offdie_b() . . . 69
9.8.2.10 dwarf_find_die_given_sig8() . . . 70
9.8.2.11 dwarf_get_die_infotypes_flag() . . . 71
9.9 Debugging Information Entry (DIE) content . . . 71
9.9.1 Detailed Description . . . 72
9.9.2 Function Documentation . . . 72
9.9.2.1 dwarf_die_abbrev_global_offset() . . . 73
9.9.2.2 dwarf_tag() . . . 73
9.9.2.3 dwarf_dieoffset() . . . 74
9.9.2.4 dwarf_debug_addr_index_to_addr() . . . 74
9.9.2.5 dwarf_addr_form_is_indexed() . . . 74
9.9.2.6 dwarf_CU_dieoffset_given_die() . . . 75
9.9.2.7 dwarf_get_cu_die_offset_given_cu_header_offset_b() . . . 75
9.9.2.8 dwarf_die_CU_offset() . . . 76
9.9.2.9 dwarf_die_CU_offset_range() . . . 76
9.9.2.10 dwarf_attr() . . . 77
9.9.2.11 dwarf_die_text() . . . 77
9.9.2.12 dwarf_diename() . . . 78
9.9.2.13 dwarf_die_abbrev_code() . . . 78
9.9.2.14 dwarf_die_abbrev_children_flag() . . . 79
9.9.2.15 dwarf_validate_die_sibling() . . . 79
9.9.2.16 dwarf_hasattr() . . . 80
9.9.2.17 dwarf_offset_list() . . . 80
9.9.2.18 dwarf_get_die_address_size() . . . 81
9.9.2.19 dwarf_die_offsets() . . . 81
9.9.2.20 dwarf_get_version_of_die() . . . 82
9.9.2.21 dwarf_lowpc() . . . 82
9.9.2.22 dwarf_highpc_b() . . . 83
9.9.2.23 dwarf_dietype_offset() . . . 83
9.9.2.24 dwarf_bytesize() . . . 84
9.9.2.25 dwarf_bitsize() . . . 84
9.9.2.26 dwarf_bitoffset() . . . 85
9.9.2.27 dwarf_srclang() . . . 85
9.9.2.28 dwarf_arrayorder() . . . 86
9.10 DIE Attribute and Attribute-Form Details . . . 86
9.10.1 Detailed Description . . . 88
9.10.2 Function Documentation . . . 88
9.10.2.1 dwarf_attrlist() . . . 88
9.10.2.2 dwarf_hasform() . . . 89

9.10.2.3 dwarf_whatform() . . . 89
9.10.2.4 dwarf_whatform_direct() . . . 89
9.10.2.5 dwarf_whatattr() . . . 90
9.10.2.6 dwarf_formref() . . . 90
9.10.2.7 dwarf_global_formref_b() . . . 91
9.10.2.8 dwarf_global_formref() . . . 92
9.10.2.9 dwarf_formsig8() . . . 92
9.10.2.10 dwarf_formsig8_const() . . . 92
9.10.2.11 dwarf_formaddr() . . . 93
9.10.2.12 dwarf_get_debug_addr_index() . . . 93
9.10.2.13 dwarf_formflag() . . . 94
9.10.2.14 dwarf_formudata() . . . 94
9.10.2.15 dwarf_formsdata() . . . 95
9.10.2.16 dwarf_formdata16() . . . 95
9.10.2.17 dwarf_formblock() . . . 96
9.10.2.18 dwarf_formstring() . . . 96
9.10.2.19 dwarf_get_debug_str_index() . . . 97
9.10.2.20 dwarf_formexprloc() . . . 97
9.10.2.21 dwarf_get_form_class() . . . 98
9.10.2.22 dwarf_attr_offset() . . . 98
9.10.2.23 dwarf_uncompress_integer_block_a() . . . 99
9.10.2.24 dwarf_dealloc_uncompressed_block() . . . 99
9.10.2.25 dwarf_convert_to_global_offset() . . . 99
9.10.2.26 dwarf_dealloc_attribute() . . . 100
9.10.2.27 dwarf_discr_list() . . . 100
9.10.2.28 dwarf_discr_entry_u() . . . 101
9.10.2.29 dwarf_discr_entry_s() . . . 101
9.11 Line Table For a CU . . . 102
9.11.1 Detailed Description . . . 103
9.11.2 Function Documentation . . . 103
9.11.2.1 dwarf_srcfiles() . . . 104
9.11.2.2 dwarf_srclines_b() . . . 105
9.11.2.3 dwarf_srclines_from_linecontext() . . . 105
9.11.2.4 dwarf_srclines_two_level_from_linecontext() . . . 106
9.11.2.5 dwarf_srclines_dealloc_b() . . . 106
9.11.2.6 dwarf_srclines_table_offset() . . . 107
9.11.2.7 dwarf_srclines_comp_dir() . . . 107
9.11.2.8 dwarf_srclines_subprog_count() . . . 108
9.11.2.9 dwarf_srclines_subprog_data() . . . 108
9.11.2.10 dwarf_srclines_files_indexes() . . . 109
9.11.2.11 dwarf_srclines_files_data_b() . . . 109
9.11.2.12 dwarf_srclines_include_dir_count() . . . 110

9.11.2.13 dwarf_srclines_include_dir_data() . . . 111
9.11.2.14 dwarf_srclines_version() . . . 111
9.11.2.15 dwarf_linebeginstatement() . . . 112
9.11.2.16 dwarf_lineendsequence() . . . 112
9.11.2.17 dwarf_lineno() . . . 113
9.11.2.18 dwarf_line_srcfileno() . . . 113
9.11.2.19 dwarf_line_is_addr_set() . . . 114
9.11.2.20 dwarf_lineaddr() . . . 114
9.11.2.21 dwarf_lineoff_b() . . . 114
9.11.2.22 dwarf_linesrc() . . . 115
9.11.2.23 dwarf_lineblock() . . . 115
9.11.2.24 dwarf_prologue_end_etc() . . . 116
9.11.2.25 dwarf_check_lineheader_b() . . . 116
9.11.2.26 dwarf_print_lines() . . . 117
9.11.2.27 dwarf_register_printf_callback() . . . 118
9.12 Ranges: code addresses in DWARF3-4 . . . 118
9.12.1 Detailed Description . . . 118
9.12.2 Function Documentation . . . 119
9.12.2.1 dwarf_get_ranges_b() . . . 119
9.12.2.2 dwarf_dealloc_ranges() . . . 119
9.13 Rnglists: code addresses in DWARF5 . . . 120
9.13.1 Detailed Description . . . 121
9.13.2 Function Documentation . . . 121
9.13.2.1 dwarf_rnglists_get_rle_head() . . . 121
9.13.2.2 dwarf_get_rnglists_entry_fields_a() . . . 121
9.13.2.3 dwarf_dealloc_rnglists_head() . . . 122
9.13.2.4 dwarf_load_rnglists() . . . 123
9.13.2.5 dwarf_get_rnglist_offset_index_value() . . . 123
9.13.2.6 dwarf_get_rnglist_head_basics() . . . 125
9.13.2.7 dwarf_get_rnglist_context_basics() . . . 125
9.13.2.8 dwarf_get_rnglist_rle() . . . 126
9.14 Locations of data: DWARF2-DWARF5 . . . 126
9.14.1 Detailed Description . . . 128
9.14.2 Function Documentation . . . 128
9.14.2.1 dwarf_get_loclist_c() . . . 128
9.14.2.2 dwarf_get_loclist_head_kind() . . . 128
9.14.2.3 dwarf_get_locdesc_entry_d() . . . 129
9.14.2.4 dwarf_get_location_op_value_c() . . . 130
9.14.2.5 dwarf_loclist_from_expr_c() . . . 130
9.14.2.6 dwarf_dealloc_loc_head_c() . . . 131
9.14.2.7 dwarf_load_loclists() . . . 131
9.14.2.8 dwarf_get_loclist_offset_index_value() . . . 132

9.14.2.9 dwarf_get_loclist_head_basics() . . . 132
9.14.2.10 dwarf_get_loclist_context_basics() . . . 133
9.14.2.11 dwarf_get_loclist_lle() . . . 133
9.15 .debug_addr access: DWARF5 . . . 134
9.15.1 Detailed Description . . . 134
9.15.2 Function Documentation . . . 134
9.15.2.1 dwarf_debug_addr_table() . . . 135
9.15.2.2 dwarf_debug_addr_by_index() . . . 135
9.15.2.3 dwarf_dealloc_debug_addr_table() . . . 136
9.16 Macro Access: DWARF5 . . . 136
9.16.1 Detailed Description . . . 137
9.16.2 Function Documentation . . . 137
9.16.2.1 dwarf_get_macro_context() . . . 137
9.16.2.2 dwarf_get_macro_context_by_offset() . . . 138
9.16.2.3 dwarf_macro_context_total_length() . . . 139
9.16.2.4 dwarf_dealloc_macro_context() . . . 139
9.16.2.5 dwarf_macro_context_head() . . . 139
9.16.2.6 dwarf_macro_operands_table() . . . 140
9.16.2.7 dwarf_get_macro_op() . . . 140
9.16.2.8 dwarf_get_macro_defundef() . . . 141
9.16.2.9 dwarf_get_macro_startend_file() . . . 142
9.16.2.10 dwarf_get_macro_import() . . . 142
9.17 Macro Access: DWARF2-4 . . . 143
9.17.1 Detailed Description . . . 143
9.17.2 Function Documentation . . . 143
9.17.2.1 dwarf_find_macro_value_start() . . . 143
9.17.2.2 dwarf_get_macro_details() . . . 144
9.18 Stack Frame Access . . . 144
9.18.1 Detailed Description . . . 147
9.18.2 Function Documentation . . . 147
9.18.2.1 dwarf_get_fde_list() . . . 147
9.18.2.2 dwarf_get_fde_list_eh() . . . 147
9.18.2.3 dwarf_dealloc_fde_cie_list() . . . 148
9.18.2.4 dwarf_get_fde_range() . . . 148
9.18.2.5 dwarf_get_fde_exception_info() . . . 149
9.18.2.6 dwarf_get_cie_of_fde() . . . 149
9.18.2.7 dwarf_get_cie_info_b() . . . 150
9.18.2.8 dwarf_get_cie_index() . . . 150
9.18.2.9 dwarf_get_fde_instr_bytes() . . . 151
9.18.2.10 dwarf_get_fde_info_for_all_regs3_b() . . . 151
9.18.2.11 dwarf_get_fde_info_for_all_regs3() . . . 152
9.18.2.12 dwarf_get_fde_info_for_reg3_c() . . . 152

9.18.2.13 dwarf_get_fde_info_for_reg3_b() . . . 153
9.18.2.14 dwarf_get_fde_info_for_cfa_reg3_c() . . . 154
9.18.2.15 dwarf_get_fde_info_for_cfa_reg3_b() . . . 155
9.18.2.16 dwarf_get_fde_for_die() . . . 155
9.18.2.17 dwarf_get_fde_n() . . . 155
9.18.2.18 dwarf_get_fde_at_pc() . . . 156
9.18.2.19 dwarf_get_cie_augmentation_data() . . . 156
9.18.2.20 dwarf_get_fde_augmentation_data() . . . 157
9.18.2.21 dwarf_expand_frame_instructions() . . . 157
9.18.2.22 dwarf_get_frame_instruction() . . . 158
9.18.2.23 dwarf_get_frame_instruction_a() . . . 160
9.18.2.24 dwarf_dealloc_frame_instr_head() . . . 160
9.18.2.25 dwarf_fde_section_offset() . . . 160
9.18.2.26 dwarf_cie_section_offset() . . . 161
9.18.2.27 dwarf_set_frame_rule_table_size() . . . 161
9.18.2.28 dwarf_set_frame_rule_initial_value() . . . 162
9.18.2.29 dwarf_set_frame_cfa_value() . . . 162
9.18.2.30 dwarf_set_frame_same_value() . . . 162
9.18.2.31 dwarf_set_frame_undefined_value() . . . 163
9.19 Abbreviations Section Details . . . 163
9.19.1 Detailed Description . . . 164
9.19.2 Function Documentation . . . 164
9.19.2.1 dwarf_get_abbrev() . . . 164
9.19.2.2 dwarf_get_abbrev_tag() . . . 165
9.19.2.3 dwarf_get_abbrev_code() . . . 165
9.19.2.4 dwarf_get_abbrev_children_flag() . . . 165
9.19.2.5 dwarf_get_abbrev_entry_b() . . . 166
9.20 String Section .debug_str Details . . . 167
9.20.1 Detailed Description . . . 167
9.20.2 Function Documentation . . . 167
9.20.2.1 dwarf_get_str() . . . 167
9.21 Str_Offsets section details . . . 168
9.21.1 Detailed Description . . . 168
9.21.2 Function Documentation . . . 168
9.21.2.1 dwarf_open_str_offsets_table_access() . . . 168
9.21.2.2 dwarf_close_str_offsets_table_access() . . . 169
9.21.2.3 dwarf_next_str_offsets_table() . . . 169
9.21.2.4 dwarf_str_offsets_value_by_index() . . . 170
9.21.2.5 dwarf_str_offsets_statistics() . . . 170
9.22 Dwarf_Error Functions . . . 171
9.22.1 Detailed Description . . . 171
9.22.2 Function Documentation . . . 171

9.22.2.1 dwarf_errno() . . . 171
9.22.2.2 dwarf_errmsg() . . . 172
9.22.2.3 dwarf_errmsg_by_number() . . . 172
9.22.2.4 dwarf_error_creation() . . . 172
9.22.2.5 dwarf_dealloc_error() . . . 173
9.23 Generic dwarf_dealloc Function . . . 173
9.23.1 Detailed Description . . . 173
9.23.2 Function Documentation . . . 174
9.23.2.1 dwarf_dealloc() . . . 174
9.24 Access to Section .debug_sup . . . 174
9.24.1 Detailed Description . . . 175
9.24.2 Function Documentation . . . 175
9.24.2.1 dwarf_get_debug_sup() . . . 175
9.25 Fast Access to .debug_names DWARF5 . . . 175
9.25.1 Detailed Description . . . 176
9.25.2 Function Documentation . . . 176
9.25.2.1 dwarf_dnames_header() . . . 176
9.25.2.2 dwarf_dealloc_dnames() . . . 177
9.25.2.3 dwarf_dnames_abbrevtable() . . . 177
9.25.2.4 dwarf_dnames_sizes() . . . 178
9.25.2.5 dwarf_dnames_offsets() . . . 179
9.25.2.6 dwarf_dnames_cu_table() . . . 179
9.25.2.7 dwarf_dnames_bucket() . . . 180
9.25.2.8 dwarf_dnames_name() . . . 180
9.25.2.9 dwarf_dnames_entrypool() . . . 181
9.25.2.10 dwarf_dnames_entrypool_values() . . . 182
9.26 Fast Access to a CU given a code address . . . 183
9.26.1 Detailed Description . . . 183
9.26.2 Function Documentation . . . 183
9.26.2.1 dwarf_get_aranges() . . . 184
9.26.2.2 dwarf_get_arange() . . . 184
9.26.2.3 dwarf_get_cu_die_offset() . . . 185
9.26.2.4 dwarf_get_arange_cu_header_offset() . . . 185
9.26.2.5 dwarf_get_arange_info_b() . . . 186
9.27 Fast Access to .debug_pubnames and more. . . . 186
9.27.1 Detailed Description . . . 187
9.27.2 Function Documentation . . . 187
9.27.2.1 dwarf_get_globals() . . . 187
9.27.2.2 dwarf_globals_by_type() . . . 188
9.27.2.3 dwarf_globals_dealloc() . . . 188
9.27.2.4 dwarf_globname() . . . 189
9.27.2.5 dwarf_global_die_offset() . . . 189

9.27.2.6 dwarf_global_cu_offset() . . . 190
9.27.2.7 dwarf_global_name_offsets() . . . 190
9.27.2.8 dwarf_global_tag_number() . . . 191
9.27.2.9 dwarf_get_globals_header() . . . 191
9.27.2.10 dwarf_return_empty_pubnames() . . . 191
9.28 Fast Access to GNU .debug_gnu_pubnames . . . 192
9.28.1 Detailed Description . . . 192
9.28.2 Function Documentation . . . 192
9.28.2.1 dwarf_get_gnu_index_head() . . . 192
9.28.2.2 dwarf_gnu_index_dealloc() . . . 193
9.28.2.3 dwarf_get_gnu_index_block() . . . 193
9.28.2.4 dwarf_get_gnu_index_block_entry() . . . 194
9.29 Fast Access to Gdb Index . . . 195
9.29.1 Detailed Description . . . 196
9.29.2 Function Documentation . . . 196
9.29.2.1 dwarf_gdbindex_header() . . . 196
9.29.2.2 dwarf_dealloc_gdbindex() . . . 197
9.29.2.3 dwarf_gdbindex_culist_array() . . . 197
9.29.2.4 dwarf_gdbindex_culist_entry() . . . 197
9.29.2.5 dwarf_gdbindex_types_culist_array() . . . 198
9.29.2.6 dwarf_gdbindex_types_culist_entry() . . . 198
9.29.2.7 dwarf_gdbindex_addressarea() . . . 199
9.29.2.8 dwarf_gdbindex_addressarea_entry() . . . 199
9.29.2.9 dwarf_gdbindex_symboltable_array() . . . 201
9.29.2.10 dwarf_gdbindex_symboltable_entry() . . . 201
9.29.2.11 dwarf_gdbindex_cuvector_length() . . . 202
9.29.2.12 dwarf_gdbindex_cuvector_inner_attributes() . . . 202
9.29.2.13 dwarf_gdbindex_cuvector_instance_expand_value() . . . 204
9.29.2.14 dwarf_gdbindex_string_by_offset() . . . 204
9.30 Fast Access to Split Dwarf (Debug Fission) . . . 205
9.30.1 Detailed Description . . . 206
9.30.2 Function Documentation . . . 206
9.30.2.1 dwarf_get_xu_index_header() . . . 206
9.30.2.2 dwarf_dealloc_xu_header() . . . 206
9.30.2.3 dwarf_get_xu_index_section_type() . . . 207
9.30.2.4 dwarf_get_xu_hash_entry() . . . 207
9.30.2.5 dwarf_get_xu_section_names() . . . 208
9.30.2.6 dwarf_get_xu_section_offset() . . . 208
9.30.2.7 dwarf_get_debugfission_for_die() . . . 209
9.30.2.8 dwarf_get_debugfission_for_key() . . . 209
9.31 Access GNU .gnu_debuglink, build-id. . . . 211
9.31.1 Detailed Description . . . 211

9.31.2 Function Documentation . . . 211
9.31.2.1 dwarf_gnu_debuglink() . . . 212
9.31.2.2 dwarf_suppress_debuglink_crc() . . . 213
9.31.2.3 dwarf_add_debuglink_global_path() . . . 214
9.31.2.4 dwarf_crc32() . . . 214
9.31.2.5 dwarf_basic_crc32() . . . 215
9.32 Harmless Error recording . . . 215
9.32.1 Detailed Description . . . 216
9.32.2 Function Documentation . . . 216
9.32.2.1 dwarf_get_harmless_error_list() . . . 216
9.32.2.2 dwarf_set_harmless_error_list_size() . . . 216
9.32.2.3 dwarf_insert_harmless_error() . . . 217
9.33 Names DW_TAG_member etc as strings . . . 217
9.33.1 Detailed Description . . . 219
9.33.2 Function Documentation . . . 219
9.33.2.1 dwarf_get_GNUIKIND_name() . . . 219
9.33.2.2 dwarf_get_EH_name() . . . 220
9.33.2.3 dwarf_get_FRAME_name() . . . 220
9.33.2.4 dwarf_get_GNUIVIS_name() . . . 220
9.33.2.5 dwarf_get_LLEX_name() . . . 220
9.33.2.6 dwarf_get_MACINFO_name() . . . 220
9.33.2.7 dwarf_get_MACRO_name() . . . 221
9.33.2.8 dwarf_get_FORM_CLASS_name() . . . 221
9.34 Object Sections Data . . . 221
9.34.1 Detailed Description . . . 222
9.34.2 Function Documentation . . . 223
9.34.2.1 dwarf_get_die_section_name() . . . 223
9.34.2.2 dwarf_get_die_section_name_b() . . . 223
9.34.2.3 dwarf_get_real_section_name() . . . 224
9.34.2.4 dwarf_get_frame_section_name() . . . 224
9.34.2.5 dwarf_get_frame_section_name_eh_gnu() . . . 225
9.34.2.6 dwarf_get_offset_size() . . . 225
9.34.2.7 dwarf_get_address_size() . . . 225
9.34.2.8 dwarf_get_line_section_name_from_die() . . . 225
9.34.2.9 dwarf_get_section_info_by_name_a() . . . 226
9.34.2.10 dwarf_get_section_info_by_name() . . . 226
9.34.2.11 dwarf_get_section_info_by_index_a() . . . 227
9.34.2.12 dwarf_get_section_info_by_index() . . . 228
9.34.2.13 dwarf_machine_architecture() . . . 228
9.34.2.14 dwarf_get_section_max_offsets_d() . . . 229
9.35 Section Groups Objectfile Data . . . 231
9.35.1 Detailed Description . . . 231

9.35.2 Function Documentation . . . 231
9.35.2.1 dwarf_sec_group_sizes() . . . 231
9.35.2.2 dwarf_sec_group_map() . . . 232
9.36 LEB Encode and Decode . . . 233
9.36.1 Detailed Description . . . 233
9.37 Miscellaneous Functions . . . 233
9.37.1 Detailed Description . . . 234
9.37.2 Function Documentation . . . 234
9.37.2.1 dwarf_package_version() . . . 234
9.37.2.2 dwarf_set_stringcheck() . . . 234
9.37.2.3 dwarf_set_reloc_application() . . . 234
9.37.2.4 dwarf_record_cmdline_options() . . . 235
9.37.2.5 dwarf_set_de_alloc_flag() . . . 235
9.37.2.6 dwarf_set_default_address_size() . . . 236
9.37.2.7 dwarf_get_universalbinary_count() . . . 236
9.37.3 Variable Documentation . . . 237
9.37.3.1 dwarf_get_endian_copy_function . . . 237
9.38 Determine Object Type of a File . . . 237
9.38.1 Detailed Description . . . 238
9.39 Using dwarf_init_path() . . . 238
9.40 Using dwarf_init_path_dl() . . . 238
9.41 Using dwarf_attrlist() . . . 239
9.42 Attaching a tied dbg . . . 240
9.43 Detaching a tied dbg . . . 240
9.44 Examining Section Group data . . . 241
9.45 Using dwarf_siblingofb() . . . 242
9.46 Using dwarf_child() . . . 242
9.47 using dwarf_validate_die_sibling . . . 243
9.48 Example walking CUs(e) . . . 244
9.49 Example walking CUs(d) . . . 246
9.50 Using dwarf_offdie_b() . . . 247
9.51 Using dwarf_offset_given_die() . . . 248
9.52 Using dwarf_attrlist() . . . 248
9.53 Using dwarf_offset_list() . . . 248
9.54 Documenting Form_Block . . . 249
9.55 Using dwarf_discr_list() . . . 250
9.56 Location/expression access . . . 251
9.57 Reading a location expression . . . 252
9.58 Using dwarf_srclines_b() . . . 253
9.59 Using dwarf_srclines_b() and linecontext . . . 256
9.60 Using dwarf_srcfiles() . . . 256
9.61 Using dwarf_get_globals() . . . 257

9.62 Using dwarf_globals_by_type() . . . 257
9.63 Reading .debug_weaknames (nonstandard) . . . 258
9.64 Reading .debug_funcnames (nonstandard) . . . 258
9.65 Reading .debug_types (nonstandard) . . . 259
9.66 Reading .debug_varnames data (nonstandard) . . . 259
9.67 Reading .debug_macinfo (DWARF2-4) . . . 263
9.68 Extracting fde, cie lists. . . . 264
9.69 Reading the .eh_frame section . . . 264
9.70 Using dwarf_expand_frame_instructions . . . 265
9.71 Reading string offsets section data . . . 266
9.72 Reading an aranges section . . . 267
9.73 Example getting .debug_ranges data . . . 268
9.74 Reading gdbindex data . . . 268
9.75 Reading gdbindex addressarea . . . 269
9.76 Reading the gdbindex symbol table . . . 270
9.77 Reading cu and tu Debug Fission data . . . 271
9.78 Reading Split Dwarf (Debug Fission) hash slots . . . 271
9.79 Reading high pc from a DIE. . . . 272
9.80 Reading Split Dwarf (Debug Fission) data . . . 272
9.81 Retrieving tag,attribute,etc names . . . 273
9.82 Using GNU debuglink data . . . 273
9.83 Accesing accessing raw rnglist . . . 274
9.84 Accessing a rnglists section . . . 275
9.85 Demonstrating reading DWARF without a file. . . . 276
9.86 A simple report on section groups. . . . 281

10 Data Structure Documentation **285**

10.1 Dwarf_Block_s Struct Reference . . . 285
10.2 Dwarf_Cmdline_Options_s Struct Reference . . . 285
10.3 Dwarf_Debug_Fission_Per_CU_s Struct Reference . . . 285
10.4 Dwarf_Form_Data16_s Struct Reference . . . 286
10.5 Dwarf_Macro_Details_s Struct Reference . . . 286
10.5.1 Detailed Description . . . 286
10.6 Dwarf_Obj_Access_Interface_a_s Struct Reference . . . 286
10.7 Dwarf_Obj_Access_Methods_a_s Struct Reference . . . 287
10.7.1 Detailed Description . . . 287
10.8 Dwarf_Obj_Access_Section_a_s Struct Reference . . . 287
10.9 Dwarf_Printf_Callback_Info_s Struct Reference . . . 288
10.10 Dwarf_Ranges_s Struct Reference . . . 288
10.11 Dwarf_Regtable3_s Struct Reference . . . 288
10.12 Dwarf_Regtable_Entry3_s Struct Reference . . . 289
10.13 Dwarf_Sig8_s Struct Reference . . . 289

11 File Documentation **291**

11.1 /home/davea/dwarf/code/src/bin/dwarfexample/jitreader.c File Reference 291

11.2 /home/davea/dwarf/code/src/bin/dwarfexample/showsectiongroups.c File Reference 291

Index **293**

Chapter 1

A Consumer Library Interface to DWARF

Author

David Anderson

Date

2023-12-08 v0.9.0

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-list (at) prevanders with final characters .org) Sorry about the simple obfuscation to keep bots away. It's actually a simple email address, not a list.

Thanks in advance for any suggestions.

1.2 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 document 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.

1.3 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.

1.4 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).

A) The recommmended 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. If DW_DLV_NO_ENTRY or DW_DLV_OK is returned the error argument is ignored 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.

1.4.1 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 is the value you passed in as dw_errarg, and can be anything. 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. The DW_DLV_ERROR code is returned from *libdwarf* and your code can do what it likes with the error situation.

```
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.

1.4.2 Error Handling Everywhere

So let us examine a case where anything could happen. And here we are taking the **recommeded 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_b(die,&newdie,error);
    if (res != DW_DLV_OK) {
        return res;
    }
    /*  Do something with newdie. */
    dwarf_dealloc_die(newdie);
    newdie = 0;
}
```

1.4.2.1 DW_DLV_OK

When res == DW_DLV_OK newdie is a valid pointer pointer and when appropriate we should do dwarf_dealloc_↩ die(newdie).

1.4.2.2 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. The exact meaning of course depends on the function called.

1.4.2.3 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.

If it's a decently large program then you want to free any local memory and return res. If a small and unimportant program print something and exit (likely leaking memory).

If you want to discard the error report from the dwarf_siblingof() 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 is not usually called), any dwarf_dealloc() that is needed will happen automatically when you call dwarf_finish().

1.4.2.4 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 or -fsanitize to ensure your code really does the extra dwarf_dealloc calls needed since when using dwarf_set_de_alloc_flag(0) dwarf_finish() only does limited cleanup.

1.5 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_d() and within a CU initiate a DIE scan using dwarf_siblingof_b(). There is currently no provision for a second 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

See also

examplecuhdr 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.

1.6 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 code addresses to source file locations.

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

1.7 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 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. But if there is some random data somewhere outside of referenced areas the reader function may fail, returning DW_DLV_ERROR. Such an error is neither a compiler bug nor a *libdwarf* bug.

1.8 Special Frame Registers

In dealing with .debug_frame or .eh_frame there are a few related 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.

Keep in mind that register values correspond to columns in the theoretical fully complete 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.

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 member rt3_cfa_rule or function dwarf_get_fde_info_for_cfa_reg3_b .

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
```

1.9 .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	main function
.debug_pubnames	Dwarf_Global	DWARF2-DWARF4	dwarf_get_globals
.debug_pubtypes	Dwarf_Type	DWARF3,DWARF4	dwarf_get_pubtypes

The following four were defined in SGI/IRIX compilers in the 1990s but never part of the DWARF standard.

It not likely you will encounter these.

.debug_funcs	Dwarf_Func	None	dwarf_get_funcs
.debug_typenames	Dwarf_Type	None	dwarf_get_types
.debug_vars	Dwarf_Var	None	dwarf_get_vars
.debug_weaks	Dwarf_Weak	None	dwarf_get_weaks

1.10 Reading DWARF with no object file present

This most commonly happens with just-in-time compilation, and someone working on the code wants do 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.

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_entrysize;
};
```

Dwarf_Obj_Access_Section_a: Your implementation of a **om_get_section_info** must fill in a few fields (leaving most zero) for *libdwarf*. The fields here are standard Elf, but for most you can just use the value zero. We assume here you will not be doing relocations at runtime.

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.

Dwarf_Obj_Access_Methods_a_s: The functions we need to access object data from *libdwarf* 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's not required that one handles multiple such in a single *libdwarf* 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.

1.11 Section Groups. Debug Fission. COMDAT groups

A typical executable or shared object is unlikely to have any section groups, and in that case what follows is irrelevant and unimportant.

COMDAT groups enable compilers and linkers to work together to eliminate blocks of duplicate DWARF and duplicate CODE.

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 (which also applies to shared objects).

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

To name such groups (defined later here) we add the following defines to libdwarf.h (the 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 *libdwarf* 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 *libdwarf* will not reveal contents of the other groups. One must pass in another groupnumber to dwarf_init_path, meaning init a new Dwarf_Debug, to get *libdwarf* to access that group.

When opening a Dwarf_Debug the following applies:

If DW_GROUPNUMBER_ANY is passed in *libdwarf* 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 *libdwarf* will chose DW_GROUPNUMBER_BASE.

If DW_GROUPNUMBER_BASE is passed in *libdwarf* 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 *libdwarf* 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 *libdwarf* accepts it, whether any sections corresponding to that groupnumber exist or not.

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.

1.12 Details on separate DWARF object access

There are, at present, two distinct approaches in use to put DWARF information into separate objects to significantly shrink the size of the executable.

One is Macos dSYM. It's a convention of placing the DWARF-containing object in a subdirectory tree.

The other is GNU debuglink and GNU debug_id. These are two distinct ways 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 object and report on the DWARF there.

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 are 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's 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.

1.13 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.
- Then you must add -lz -lzstd to the link line of the build of your application.

When compiling to link against a shared library libdwarf.so or libdwarf.dll you must not define LIBDWARF_STATIC.

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

1.14 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.

1.15 Recent Changes

We list these with newest first.

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 forseeable future. However it would be easy to misuse as the requirement that dwarf_siblingof_b() be called immediately after a successfull 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 1990's 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's now possible to run the build sanity tests in all three build mechanisms (configure,cmake,meson) on linux, Mac↩ OS, 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

libdwarf 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.

libdwarf 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, *libdwarf* 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 libdwarf-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 libdwarf.pdf with a libdwarf.pdf generated by doxygen and latex.

Added support for the meson build system.

Updated an include in libdwarfp source files. Improved doxygen documentation of *libdwarf*. 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_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_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_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_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
    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.
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.
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 libdwarf
might read.  Returns a value that is a sanity check on
offsets 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 n 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 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 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 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

Module Index

6.1 Modules

Here is a list of all modules:

Basic Library Datatypes Group . . . 35
Enumerators with various purposes . . . 37
Defined and Opaque Structs . . . 38
Default stack frame #defines . . . 45
DW_DLA alloc/dealloc typename&number . . . 46
DW_DLE Dwarf_Error numbers . . . 47
Libdwarf Initialization Functions . . . 56
Compilation Unit (CU) Access . . . 63
Debugging Information Entry (DIE) content . . . 71
DIE Attribute and Attribute-Form Details . . . 86
Line Table For a CU . . . 102
Ranges: code addresses in DWARF3-4 . . . 118
Rnglists: code addresses in DWARF5 . . . 120
Locations of data: DWARF2-DWARF5 . . . 126
.debug_addr access: DWARF5 . . . 134
Macro Access: DWARF5 . . . 136
Macro Access: DWARF2-4 . . . 143
Stack Frame Access . . . 144
Abbreviations Section Details . . . 163
String Section .debug_str Details . . . 167
Str_Offsets section details . . . 168
Dwarf_Error Functions . . . 171
Generic dwarf_dealloc Function . . . 173
Access to Section .debug_sup . . . 174
Fast Access to .debug_names DWARF5 . . . 175
Fast Access to a CU given a code address . . . 183
Fast Access to .debug_pubnames and more. . . . 186
Fast Access to GNU .debug_gnu_pubnames . . . 192
Fast Access to Gdb Index . . . 195
Fast Access to Split Dwarf (Debug Fission) . . . 205
Access GNU .gnu_debuglink, build-id. . . . 211
Harmless Error recording . . . 215
Names DW_TAG_member etc as strings . . . 217
Object Sections Data . . . 221
Section Groups Objectfile Data . . . 231

LEB Encode and Decode . . . 233
Miscellaneous Functions . . . 233
Determine Object Type of a File . . . 237
Using dwarf_init_path() . . . 238
Using dwarf_init_path_dl() . . . 238
Using dwarf_attrlist() . . . 239
Attaching a tied dbg . . . 240
Detaching a tied dbg . . . 240
Examining Section Group data . . . 241
Using dwarf_siblingofb() . . . 242
Using dwarf_child() . . . 242
using dwarf_validate_die_sibling . . . 243
Example walking CUs(e) . . . 244
Example walking CUs(d) . . . 246
Using dwarf_offdie_b() . . . 247
Using dwarf_offset_given_die() . . . 248
Using dwarf_attrlist() . . . 248
Using dwarf_offset_list() . . . 248
Documenting Form_Block . . . 249
Using dwarf_discr_list() . . . 250
Location/expression access . . . 251
Reading a location expression . . . 252
Using dwarf_srclines_b() . . . 253
Using dwarf_srclines_b() and linecontext . . . 256
Using dwarf_srcfiles() . . . 256
Using dwarf_get_globals() . . . 257
Using dwarf_globals_by_type() . . . 257
Reading .debug_weaknames (nonstandard) . . . 258
Reading .debug_funcnames (nonstandard) . . . 258
Reading .debug_types (nonstandard) . . . 259
Reading .debug_varnames data (nonstandard) . . . 259
Reading .debug_macinfo (DWARF2-4) . . . 263
Extracting fde, cie lists. . . . 264
Reading the .eh_frame section . . . 264
Using dwarf_expand_frame_instructions . . . 265
Reading string offsets section data . . . 266
Reading an aranges section . . . 267
Example getting .debug_ranges data . . . 268
Reading gdbindex data . . . 268
Reading gdbindex addressarea . . . 269
Reading the gdbindex symbol table . . . 270
Reading cu and tu Debug Fission data . . . 271
Reading Split Dwarf (Debug Fission) hash slots . . . 271
Reading high pc from a DIE. . . . 272
Reading Split Dwarf (Debug Fission) data . . . 272
Retrieving tag,attribute,etc names . . . 273
Using GNU debuglink data . . . 273
Accesing accessing raw rnglist . . . 274
Accessing a rnglists section . . . 275
Demonstrating reading DWARF without a file. . . . 276
A simple report on section groups. . . . 281

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

Dwarf_Block_s 285
Dwarf_Cmdline_Options_s 285
Dwarf_Debug_Fission_Per_CU_s 285
Dwarf_Form_Data16_s 286
Dwarf_Macro_Details_s 286
Dwarf_Obj_Access_Interface_a_s 286
Dwarf_Obj_Access_Methods_a_s 287
Dwarf_Obj_Access_Section_a_s 287
Dwarf_Printf_Callback_Info_s 288
Dwarf_Ranges_s 288
Dwarf_Regtable3_s 288
Dwarf_Regtable_Entry3_s 289
Dwarf_Sig8_s 289

Chapter 8

File Index

8.1 File List

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

checkexamples.c 27
/home/davea/dwarf/code/src/bin/dwarfexample/jitreader.c 291
/home/davea/dwarf/code/src/bin/dwarfexample/showsectiongroups.c 291
/home/davea/dwarf/code/src/lib/libdwarf/dwarf.h 23
/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h 25

Chapter 9

Module 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_Unsigned

`Dwarf_Unsigned`

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

9.1.2.2 Dwarf_Signed

`Dwarf_Signed`

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

9.1.2.3 Dwarf_Off

Dwarf_Off

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

9.1.2.4 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.

9.1.2.5 Dwarf_Bool

Dwarf_Bool

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

9.1.2.6 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).

9.1.2.7 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_Ptr

Dwarf_Ptr

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

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_Ranges_Entry_Type

enum Dwarf_Ranges_Entry_Type

The dwr_addr1/addr2 data is either an offset (DW_RANGES_ENTRY) or an address (dwr_addr2 in DW_RANGES_ADDRESS_SELECTION) or both are zero (DW_RANGES_END). For DWARF5 each table starts with a header followed by range list entries defined as here. Dwarf_Ranges* apply to DWARF2,3, and 4. Not to DWARF5 (the data is different and in a new DWARF5 section).

9.2.2.2 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 we 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 aways 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 aways a section offset, never an index. DW_FORM_CLASS_RNGLISTSPTR is only referenced by DW_AT_rnglists_base.

9.3 Defined and Opaque Structs

Data Structures

- 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 *, const char *)
- 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_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

9.3.1 Detailed Description

9.3.2 Typedef Documentation

9.3.2.1 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.

9.3.2.2 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

9.3.2.3 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

9.3.2.4 Dwarf_Locdesc_c

Dwarf_Locdesc_c

Provides access to Dwarf_Locdesc_c, a single location description

9.3.2.5 Dwarf_Loc_Head_c

Dwarf_Loc_Head_c

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

9.3.2.6 Dwarf_Dsc_Head

Dwarf_Dsc_Head

Access to DW_AT_discr_list array of discriminant values.

9.3.2.7 Dwarf_Frame_Instr_Head

Dwarf_Frame_Instr_Head

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

9.3.2.8 dwarf_printf_callback_function_type

dwarf_printf_callback_function_type

Used as a function pointer to a user-written callback function.

9.3.2.9 Dwarf_Cmdline_Options

Dwarf_Cmdline_Options

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.

9.3.2.10 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.

9.3.2.11 Dwarf_Ranges

Dwarf_Ranges

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

9.3.2.12 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 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.

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

9.3.2.13 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 libdwarf. 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. libdwarf does not allocate or deallocate space for the rules, you must do so. libdwarf 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: libdwarf 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.

9.3.2.14 Dwarf_Error

Dwarf_Error

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

9.3.2.15 Dwarf_Debug

Dwarf_Debug

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

9.3.2.16 Dwarf_Die

Dwarf_Die

Used to reference a DWARF Debugging Information Entry.

9.3.2.17 Dwarf_Debug_Addr_Table

`Dwarf_Debug_Addr_Table`

Used to reference a table in section .debug_addr

9.3.2.18 Dwarf_Line

`Dwarf_Line`

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

9.3.2.19 Dwarf_Global

`Dwarf_Global`

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

9.3.2.20 Dwarf_Type

`Dwarf_Type`

Used to reference a reference to an entry in the .debug_pubtypes section (as well as the SGI-only extension .debug_types).

9.3.2.21 Dwarf_Attribute

`Dwarf_Attribute`

Used to reference a Dwarf_Die attribute

9.3.2.22 Dwarf_Abbrev

`Dwarf_Abbrev`

Used to reference a Dwarf_Abbrev, though usually such are handled transparently in the library

9.3.2.23 Dwarf_Fde

`Dwarf_Fde`

Used to reference .debug_frame or .eh_frame FDE.

9.3.2.24 Dwarf_Cie

`Dwarf_Cie`

Used to reference .debug_frame or .eh_frame CIE.

9.3.2.25 Dwarf_Arange

`Dwarf_Arange`

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

9.3.2.26 Dwarf_Gdbindex

`Dwarf_Gdbindex`

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

9.3.2.27 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.

9.3.2.28 Dwarf_Line_Context

`Dwarf_Line_Context`

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

9.3.2.29 Dwarf_Macro_Context

`Dwarf_Macro_Context`

Used as the general reference to DWARF5 .debug_macro data.

9.3.2.30 Dwarf_Dnames_Head

`Dwarf_Dnames_Head`

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

9.3.2.31 Dwarf_Handler

`Dwarf_Handler`

Used in rare cases (mainly tiny programs) with dwarf_init_path() etc initialization calls.

9.3.2.32 Dwarf_Obj_Access_Interface_a

Dwarf_Obj_Access_Interface_a

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

9.3.2.33 Dwarf_Obj_Access_Methods_a

Dwarf_Obj_Access_Methods_a

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

9.3.2.34 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.

9.3.2.35 Dwarf_Rnglists_Head

Dwarf_Rnglists_Head

Used for access to a set of DWARF5 debug_rnglists entries.

9.4 Default stack frame #defines

Macros

- #define **DW_DLX_NO_EH_OFFSET** (-1LL)
- #define **DW_DLX_NO_EH_OFFSET** (-1LL)
- #define **DW_DLX_EH_OFFSET_UNAVAILABLE** (-2LL)
- #define **DW_DLX_EH_OFFSET_UNAVAILABLE** (-2LL)
- #define **DW_CIE_AUGMENTER_STRING_V0** "z"
- #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_LOCDESC** 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_LOCDESC_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 libdwarf, 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_VARNAMES_DUPLICATE** 164
- #define **DW_DLE_DEBUG_VARNAMES_NULL** 165
- #define **DW_DLE_DEBUG_VARNAMES_VERSION_ERROR** 166
- #define **DW_DLE_DEBUG_VARNAMES_LENGTH_BAD** 167
- #define **DW_DLE_VAR_NULL** 168
- #define **DW_DLE_VAR_CONTEXT_NULL** 169
- #define **DW_DLE_DEBUG_WEAKNAMES_DUPLICATE** 170
- #define **DW_DLE_DEBUG_WEAKNAMES_NULL** 171
- #define **DW_DLE_DEBUG_WEAKNAMES_VERSION_ERROR** 172
- #define **DW_DLE_DEBUG_WEAKNAMES_LENGTH_BAD** 173
- #define **DW_DLE_WEAK_NULL** 174
- #define **DW_DLE_WEAK_CONTEXT_NULL** 175
- #define **DW_DLE_LOCDESC_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_VARNAMES_ERROR** 278
- #define **DW_DLE_WEAKNAMES_ERROR** 279
- #define **DW_DLE_RELOCS_ERROR** 280
- #define **DW_DLE_ATTR_OUTSIDE_SECTION** 281
- #define **DW_DLE_FISSION_INDEX_WRONG** 282
- #define **DW_DLE_FISSION_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_FISSION_INCOMPLETE** 294
- #define **DW_DLE_FISSION_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_FISSION_ADDITION_ERROR** 341
- #define **DW_DLE_HEADER_LEN_BIGGER_THAN_SECSIZE** 342
- #define **DW_DLE_LOCEXPR_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_LOCIISTS_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_LAST 503
- #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 503
```

Note

DW_DLE_LAST MUST EQUAL LAST ERROR NUMBER

9.7 Libdwarf Initialization Functions

Functions

- 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.
- 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.

- 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.
- 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.
- 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) path This version allows specifying any number of debuglink global paths to search on for debuglink targets.
- int dwarf_finish (Dwarf_Debug dw_dbg)

  Close the initialized dw_dbg and free all data libdwarf has for this dw_dbg.
- 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.
- int dwarf_object_finish (Dwarf_Debug dw_dbg)

  Used to close the object_init dw_dbg.
- int dwarf_set_tied_dbg (Dwarf_Debug dw_basedbg, Dwarf_Debug dw_tied_dbg, Dwarf_Error *dw_error)

  Use with split dwarf.
- 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_init_path()

```
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

dw_path	Pass in the path to the object file to open.
dw_true_path_out_buffer	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.
dw_true_path_bufferlen	Pass in the length in bytes of the buffer.
dw_groupnumber	The value passed in should be DW_GROUPNUMBER_ANY unless one wishes to other than a standard group.
dw_errhand	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.
dw_errarg	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.
dw_dbg	On success, ∗dw_dbg is set to a pointer to a new Dwarf_Debug structure to be used in calls to libdwarf functions.
dw_error	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

dwarf_init_path_dl dwarf_init_b

Using dwarf_init_path()

9.7.3.2 dwarf_init_path_a()

```
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.

9.7.3.3 dwarf_init_path_dl()

```
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

dw_path	Pass in the path to the object file to open.
dw_true_path_out_buffer	Pass in NULL or the name of a string buffer.
dw_true_path_bufferlen	Pass in the length in bytes of the buffer.
dw_groupnumber	The value passed in should be DW_GROUPNUMBER_ANY unless one wishes to other than a standard group.
dw_errhand	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.
dw_errarg	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.
dw_dbg	On success, ∗dw_dbg is set to a pointer to a new Dwarf_Debug structure to be used in calls to libdwarf functions.
dw_dl_path_array	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.)
dw_dl_path_array_size	Specify the size of the dw_dl_path_array.
dw_dl_path_source	returns DW_PATHSOURCE_basic or other such value so the caller can know how the true-path was resolved.
dw_error	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()

9.7.3.4 dwarf_init_path_dl_a()

```
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.

9.7.3.5 dwarf_init_b()

```
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) path This version allows specifying any number of debuglink global paths to search on for debuglink targets.

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

Parameters

dw_fd	An open Unix/Linux/etc fd on the object file.
dw_groupnumber	The value passed in should be DW_GROUPNUMBER_ANY unless one wishes to other than a standard group.
dw_errhand	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.
dw_errarg	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.
dw_dbg	On success, *dw_dbg is set to a pointer to a new Dwarf_Debug structure to be used in calls to libdwarf functions.
dw_error	In case return is DW_DLV_ERROR dw_error is set to point to the error details.

Returns

DW_DLV_OK etc.

9.7.3.6 dwarf_finish()

```
int dwarf_finish (
            Dwarf_Debug dw_dbg )
```

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

Parameters

dw_dbg	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.

9.7.3.7 dwarf_object_init_b()

```
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.

See also

Demonstrating reading DWARF without a file.

and

See also

dw_noobject Reading DWARF not in object file

Parameters

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

9.7.3.8 dwarf_object_finish()

```
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

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

Returns

The return value DW_DLV_OK etc is pretty useless, there is not much you can do with it.

9.7.3.9 dwarf_set_tied_dbg()

```
int dwarf_set_tied_dbg (
            Dwarf_Debug dw_basedbg,
            Dwarf_Debug dw_tied_dbg,
            Dwarf_Error * dw_error )
```

Use with split dwarf.

Parameters

dw_basedbg	Pass in an open dbg, on an object file with (normally) lots of DWARF..
dw_tied_dbg	Pass in an open dbg on an executable which has minimal DWARF to save space in the executable.
dw_error	In case return is DW_DLV_ERROR dw_error is set to point to the error details.

Returns

DW_DLV_OK etc.

See also

Attaching a tied dbg

Detaching a tied dbg

9.7.3.10 dwarf_get_tied_dbg()

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

Use with split dwarf.

Given a base Dwarf_Debug this returns the tied Dwarf_Debug. Unlikely anyone uses this call as you had the tied and base dbg when calling dwarf_set_tied_dbg().

9.8 Compilation Unit (CU) Access

Functions

- 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).
- 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)
- int dwarf_siblingof_c (Dwarf_Die dw_die, Dwarf_Die *dw_return_siblingdie, Dwarf_Error *dw_error)

  Return the next sibling DIE.
- 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.

- 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.
- 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.
- void dwarf_dealloc_die (Dwarf_Die dw_die)

  Deallocate (free) a DIE.
- 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.
- 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.
- 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.
- 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_next_cu_header_e()

```
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.

The library keeps track of where it is in the object file and it knows where to find 'next'.

It returns the CU_DIE pointer through dw_cu_die;

Parameters

dw_dbg	The Dwarf_Debug of interest.
dw_is_info	Pass in TRUE if reading through .debug_info Pass in FALSE if reading through DWARF4 .debug_types.
dw_cu_die	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.
dw_cu_header_length	Returns the length of the just-read CU header.
dw_version_stamp	Returns the version number (2 to 5) of the CU header just read.
dw_abbrev_offset	Returns the .debug_abbrev offset from the the CU header just read.
dw_address_size	Returns the address size specified for this CU, usually either 4 or 8.
dw_length_size	Returns the offset size (the length of the size field from the header) specified for this CU, either 4 or 4.
dw_extension_size	If the section is standard 64bit DWARF then this value is 4. Else the value is zero.
dw_type_signature	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.
dw_typeoffset	For DW_UT_split_type or DW_UT_type this is the type offset from the CU header.
dw_next_cu_header_offset	The offset in the section of the next CU (unless there is a compiler bug this is rarely of interest).
dw_header_cu_type	Returns DW_UT_compile, or other DW_UT value.
dw_error	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)

9.8.2.2 dwarf_next_cu_header_d()

```
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).

This version will evenually be deprecated, but that won't be for years.

The library keeps track of where it is in the object file and it knows where to find 'next'.

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 here.

9.8.2.3 dwarf_siblingof_c()

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

Return the next sibling DIE.

Parameters

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

Returns

Returns DW_DLV_OK etc.

See also

Using dwarf_siblingofb()

dwarf_get_die_infotypes

9.8.2.4 dwarf_siblingof_b()

```
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.

Parameters

dw_dbg	The Dwarf_Debug one is operating on.
dw_die	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.
dw_is_info	Pass TRUE or FALSE to match the applicable dwarf_next_cu_header_d call.
dw_return_siblingdie	The DIE returned through the pointer.
dw_error	The usual error information, if any.

Returns

Returns DW_DLV_OK etc.

See also

Using dwarf_siblingofb()

dwarf_get_die_infotypes

9.8.2.5 dwarf_cu_header_basics()

```
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

dw_die	Some open Dwarf_Die.
dw_version	Returns the DWARF version: 2,3,4, or 5
dw_is_info	Returns non-zero if the CU is .debug_info. Returns zero if the CU is .debug_types (DWARF4).
dw_is_dwo	Returns ton-zero if the CU is a dwo/dwp object and zero if it is a standard object.
dw_offset_size	Returns offset size, 4 and 8 are possible.
dw_address_size	Almost always returns 4 or 8. Could be 2 in unusual circumstances.
dw_extension_size	The sum of dw_offset_size and dw_extension_size 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.
dw_signature	Returns a pointer to an 8 byte signature.
dw_offset_of_length	Returns the section offset of the initial byte of the CU.
dw_total_byte_length	Returns the total length of the CU including the length field and the content of the CU.
dw_error	The usual Dwarf_Error*.

Returns

Returns DW_DLV_OK etc.

9.8.2.6 dwarf_child()

```
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.

Parameters

dw_die	We will return the first child of this DIE.
dw_return_childdie	Returns the first child through the pointer. For subsequent dies siblings of the first, use dwarf_siblingof_b().
dw_error	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()

9.8.2.7 dwarf_dealloc_die()

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

Deallocate (free) a DIE.

Parameters

dw_die	Frees (deallocs) memory associated with this Dwarf_Die.

9.8.2.8 dwarf_die_from_hash_signature()

```
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

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

9.8.2.9 dwarf_offdie_b()

```
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

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

Returns

DW_DLV_OK means dw_returned_die was found DW_DLV_NO_ENTRY is only possible if the offset is to a null DIE, and that is very unusual. Otherwise expect DW_DLV_ERROR.

See also

Using dwarf_offdie_b()

9.8.2.10 dwarf_find_die_given_sig8()

```
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

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

Returns

Returns DW_DLV_OK etc.

9.8.2.11 dwarf_get_die_infotypes_flag()

```
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

dw_die	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.

9.9 Debugging Information Entry (DIE) content

Functions

- 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.
- int dwarf_tag (Dwarf_Die dw_die, Dwarf_Half *dw_return_tag, Dwarf_Error *dw_error)

  Get TAG value of DIE.
- int dwarf_dieoffset (Dwarf_Die dw_die, Dwarf_Off *dw_return_offset, Dwarf_Error *dw_error)

  Return the global section offset of the DIE.
- 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.
- Dwarf_Bool dwarf_addr_form_is_indexed (int dw_form)

  Informs if a DW_FORM is an indexed form.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- int dwarf_diename (Dwarf_Die dw_die, char **dw_diename, Dwarf_Error *dw_error)

Return the string from a DW_AT_name attribute.

- int dwarf_die_abbrev_code (Dwarf_Die dw_die)

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

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

  Validate a sibling DIE.
- 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.
- 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.
- 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.
- 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.
- 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.
- int dwarf_lowpc (Dwarf_Die dw_die, Dwarf_Addr *dw_returned_addr, Dwarf_Error *dw_error)

  Return the DW_AT_low_pc value.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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_die_abbrev_global_offset()

```
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

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

Returns

Returns DW_DLV_OK etc.

9.9.2.2 dwarf_tag()

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

Get TAG value of DIE.

Parameters

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

Returns

Returns DW_DLV_OK etc.

9.9.2.3 dwarf_dieoffset()

```
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

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

Returns

Returns DW_DLV_OK etc.

9.9.2.4 dwarf_debug_addr_index_to_addr()

```
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.

Parameters

dw_die	The DIE of interest
dw_index	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.
dw_return_addr	On success the address is returned through the pointer.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.5 dwarf_addr_form_is_indexed()

```
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.

9.9.2.6 dwarf_CU_dieoffset_given_die()

```
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).

See also

dwarf_get_cu_die_offset_given_cu_header_offset_b

Using dwarf_offset_given_die()

Parameters

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

Returns

Returns DW_DLV_OK etc.

9.9.2.7 dwarf_get_cu_die_offset_given_cu_header_offset_b()

```
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

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

Returns

Returns DW_DLV_OK etc.

9.9.2.8 dwarf_die_CU_offset()

```
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

Parameters

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

Returns

Returns DW_DLV_OK etc.

9.9.2.9 dwarf_die_CU_offset_range()

```
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.

Parameters

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

Returns

Returns DW_DLV_OK etc.

9.9.2.10 dwarf_attr()

```
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

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

Returns

Returns DW_DLV_OK etc.

9.9.2.11 dwarf_die_text()

```
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

dw_die	The DIE of interest.
dw_attrnum	An attribute number, for example DW_AT_name.
dw_ret_name	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 follows all of them to their string.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.12 dwarf_diename()

```
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

dw_die	The DIE of interest.
dw_diename	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.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.13 dwarf_die_abbrev_code()

```
int dwarf_die_abbrev_code (
            Dwarf_Die dw_die )
```

Return the DIE abbrev code.

The Abbrev code for a DIE is an integer assigned by the compiler within a particular CU. For .debug_names abbreviations the situation is different.

Returns the abbrev code of the die. Cannot fail.

Parameters

dw_die	The DIE of interest.

Returns

The abbrev code. of the DIE.

9.9.2.14 dwarf_die_abbrev_children_flag()

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

Return TRUE if the DIE has children.

Parameters

dw_die	A DIE.
dw_ab_has_child	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.

9.9.2.15 dwarf_validate_die_sibling()

```
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.

See also

using dwarf_validate_die_sibling

Parameters

dw_sibling	Pass in a DIE returned by dwarf_siblingof_b().
dw_offset	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.

9.9.2.16 dwarf_hasattr()

```
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

dw_die	The DIE of interest.
dw_attrnum	The attribute number we are asking about, DW_AT_name for example.
dw_returned_bool	On success is set TRUE if dw_die has dw_attrnum.
dw_error	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.

9.9.2.17 dwarf_offset_list()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_offset	A DIE offset.
dw_is_info	If TRUE says to use the offset in .debug_info. Else use the offset in .debug_types.
dw_offbuf	A pointer to an array of children DIE global [section] offsets is returned through the pointer.
dw_offcount	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.
dw_error	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()

9.9.2.18 dwarf_get_die_address_size()

```
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

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

Returns

Returns DW_DLV_OK etc.

9.9.2.19 dwarf_die_offsets()

```
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.

Parameters

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

Returns

Returns DW_DLV_OK etc.

9.9.2.20 dwarf_get_version_of_die()

```
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

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

9.9.2.21 dwarf_lowpc()

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

Return the DW_AT_low_pc value.

Parameters

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

Returns

Returns DW_DLV_OK etc.

9.9.2.22 dwarf_highpc_b()

```
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.

This is accessing the DW_AT_high_pc 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

dw_die	The DIE of interest.
dw_return_addr	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.
dw_return_form	On success returns the actual FORM for this attribute. Needed for certain cases to calculate the true dw_return_addr;
dw_return_class	On success returns the FORM CLASS for this attribute. Needed for certain cases to calculate the true dw_return_addr;
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.23 dwarf_dietype_offset()

```
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 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

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

Returns

Returns DW_DLV_OK etc.

9.9.2.24 dwarf_bytesize()

```
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

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

Returns

Returns DW_DLV_OK etc.

9.9.2.25 dwarf_bitsize()

```
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

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

Returns

Returns DW_DLV_OK etc.

9.9.2.26 dwarf_bitoffset()

```
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

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

Returns

Returns DW_DLV_OK etc.

9.9.2.27 dwarf_srclang()

```
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.

The DIE should be a CU DIE.

Parameters

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

Returns

Returns DW_DLV_OK etc.

9.9.2.28 dwarf_arrayorder()

```
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

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

Returns

Returns DW_DLV_OK etc.

9.10 DIE Attribute and Attribute-Form Details

Functions

- 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.
- 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.
- int dwarf_whatform (Dwarf_Attribute dw_attr, Dwarf_Half *dw_returned_final_form, Dwarf_Error *dw_error)

  Return the form of the Dwarf_Attribute.
- 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.
- int dwarf_whatattr (Dwarf_Attribute dw_attr, Dwarf_Half *dw_returned_attrnum, Dwarf_Error *dw_error)

  Return the attribute number of the Dwarf_Attribute.
- 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.
- 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.
- int dwarf_global_formref (Dwarf_Attribute dw_attr, Dwarf_Off *dw_return_offset, Dwarf_Error *dw_error)

Same as dwarf_global_formref_b except...

- 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.
- 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.
- 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.
- 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.
- int dwarf_formflag (Dwarf_Attribute dw_attr, Dwarf_Bool *dw_returned_bool, Dwarf_Error *dw_error)

  Return the flag value of a flag form.
- int dwarf_formudata (Dwarf_Attribute dw_attr, Dwarf_Unsigned *dw_returned_val, Dwarf_Error *dw_error)

  Return an unsigned value.
- int dwarf_formsdata (Dwarf_Attribute dw_attr, Dwarf_Signed *dw_returned_val, Dwarf_Error *dw_error)

  Return a signed value.
- 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.
- int dwarf_formblock (Dwarf_Attribute dw_attr, Dwarf_Block **dw_returned_block, Dwarf_Error *dw_error)

  Return an allocated filled-in Form_Block.
- int dwarf_formstring (Dwarf_Attribute dw_attr, char **dw_returned_string, Dwarf_Error *dw_error)

  Return a pointer to a string.
- int dwarf_get_debug_str_index (Dwarf_Attribute dw_attr, Dwarf_Unsigned *dw_return_index, Dwarf_Error *dw_error)

  Return a string index.
- 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.
- 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.
- 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.
- 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.
- void dwarf_dealloc_uncompressed_block (Dwarf_Debug dw_dbg, void *dw_value_array)

  Dealloc what dwarf_uncompress_integer_block_a allocated.
- 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.
- void dwarf_dealloc_attribute (Dwarf_Attribute dw_attr)

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

- 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.

9.10.1 Detailed Description

Access to the details of DIEs

9.10.2 Function Documentation

9.10.2.1 dwarf_attrlist()

```
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

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

9.10.2.2 dwarf_hasform()

```
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

dw_attr	The Dwarf_Attribute of interest.
dw_form	The DW_FORM you are asking about, DW_FORM_strp for example.
dw_returned_bool	The pointer passed in must be a valid non-null pointer to a Dwarf_Bool. On success, sets the value to TRUE or FALSE.
dw_error	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;

9.10.2.3 dwarf_whatform()

```
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

dw_attr	The Dwarf_Attribute of interest.
dw_returned_final_form	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.
dw_error	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;

9.10.2.4 dwarf_whatform_direct()

```
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

dw_attr	The Dwarf_Attribute of interest.
dw_returned_initial_form	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.
dw_error	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;

9.10.2.5 dwarf_whatattr()

```
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

dw_attr	The Dwarf_Attribute of interest.
dw_returned_attrnum	The attribute number of the attribute is returned through the pointer. For example, DW_AT_name
dw_error	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;

9.10.2.6 dwarf_formref()

```
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

dw_attr	The Dwarf_Attribute of interest.
dw_return_offset	Returns the CU-relative offset through the pointer.
dw_is_info	Returns a flag through the pointer. TRUE if the offset is in .debug_info, FALSE if it is in .debug_types
dw_error	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;

9.10.2.7 dwarf_global_formref_b()

```
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

dw_attr	The Dwarf_Attribute of interest.
dw_return_offset	Returns the CU-relative offset through the pointer.
dw_offset_is_info	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.
dw_error	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;

9.10.2.8 dwarf_global_formref()

```
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!

9.10.2.9 dwarf_formsig8()

```
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

dw_attr	The Dwarf_Attribute of interest.
dw_returned_sig_bytes	On success returns DW_DLV_OK and copies the 8 bytes into dw_returned_sig_bytes.
dw_error	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

9.10.2.10 dwarf_formsig8_const()

```
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

dw_attr	The Dwarf_Attribute of interest.
dw_returned_sig_bytes	On success Returns DW_DLV_OK and copies the 8 bytes into dw_returned_sig_bytes.
dw_error	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

9.10.2.11 dwarf_formaddr()

```
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

dw_attr	The Dwarf_Attribute of interest.
dw_returned_addr	On success this set through the pointer to the address in the attribute.
dw_error	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.

9.10.2.12 dwarf_get_debug_addr_index()

```
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 does have an index address FORM such as DW_FORM_addrx1 or one of the GNU address index forms.

Parameters

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

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.13 dwarf_formflag()

```
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

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

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.14 dwarf_formudata()

```
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

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

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.15 dwarf_formsdata()

```
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

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

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.16 dwarf_formdata16()

```
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

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

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.17 dwarf_formblock()

```
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

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

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.18 dwarf_formstring()

```
int dwarf_formstring (
            Dwarf_Attribute dw_attr,
            char ** dw_returned_string,
            Dwarf_Error * dw_error )
```

Return a pointer to a string.

Parameters

dw_attr	The Dwarf_Attribute of interest.
dw_returned_string	Puts a pointer to a string in the DWARF information if the FORM of the attribute is some sort of string FORM.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.10.2.19 dwarf_get_debug_str_index()

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

Return a string index.

Parameters

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

9.10.2.20 dwarf_formexprloc()

```
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

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

9.10.2.21 dwarf_get_form_class()

```
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

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

9.10.2.22 dwarf_attr_offset()

```
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

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

Returns

DW_DLV_OK if it succeeds. DW_DLV_NO_ENTRY is impossible.

9.10.2.23 dwarf_uncompress_integer_block_a()

```
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

9.10.2.24 dwarf_dealloc_uncompressed_block()

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

Dealloc what dwarf_uncompress_integer_block_a allocated.

Parameters

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

9.10.2.25 dwarf_convert_to_global_offset()

```
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

dw_attr	The attribute the local offset was extracted from.
dw_offset	The global offset of the attribute.
dw_return_offset	The returned section (global) offset.
dw_error	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).

9.10.2.26 dwarf_dealloc_attribute()

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

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

Parameters

dw_attr	The attribute to dealloc.

9.10.2.27 dwarf_discr_list()

```
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

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

Returns

DW_DLV_OK if it succeeds.

9.10.2.28 dwarf_discr_entry_u()

```
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

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

Returns

DW_DLV_OK if it succeeds.

9.10.2.29 dwarf_discr_entry_s()

```
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.

9.11 Line Table For a CU

Functions

- 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.
- 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.
- 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.
- 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.
- void dwarf_srclines_dealloc_b (Dwarf_Line_Context dw_context)

  Dealloc the memory allocated by dwarf_srclines_b.
- int dwarf_srclines_table_offset (Dwarf_Line_Context dw_context, Dwarf_Unsigned *dw_offset, Dwarf_Error *dw_error)

  Return the srclines table offset.
- 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.
- 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.
- 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.
- 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().
- 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.
- 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.
- 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.
- 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.
- int dwarf_linebeginstatement (Dwarf_Line dw_line, Dwarf_Bool *dw_returned_bool, Dwarf_Error *dw_error)

  Read Line beginstatement register.
- int dwarf_lineendsequence (Dwarf_Line dw_line, Dwarf_Bool *dw_returned_bool, Dwarf_Error *dw_error)

  Read Line endsequence register flag.

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

  Read Line line register.
- int dwarf_line_srcfileno (Dwarf_Line dw_line, Dwarf_Unsigned *dw_returned_filenum, Dwarf_Error *dw_↩ error)

  Read Line file register.
- 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.
- int dwarf_lineaddr (Dwarf_Line dw_line, Dwarf_Addr *dw_returned_addr, Dwarf_Error *dw_error)

  Return the address of the Dwarf_Line.
- int dwarf_lineoff_b (Dwarf_Line dw_line, Dwarf_Unsigned *dw_returned_lineoffset, Dwarf_Error *dw_error)

  Return a column number through the pointer.
- int dwarf_linesrc (Dwarf_Line dw_line, char **dw_returned_name, Dwarf_Error *dw_error)

  Return the file name applicable to the Dwarf_Line.
- int dwarf_lineblock (Dwarf_Line dw_line, Dwarf_Bool *dw_returned_bool, Dwarf_Error *dw_error)

  Return the basic_block line register.
- 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.
- 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.
- 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.
- int dwarf_line_subprogno (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_subprogno() 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.
- 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.
- 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.
- int dwarf_print_lines (Dwarf_Die dw_cu_die, Dwarf_Error *dw_error, int *dw_errorcount_out)

  Print line information in great detail.
- 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_srcfiles()

```
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 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** 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:**

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

dw_cu_die	The CU DIE in this CU.
dw_srcfiles	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.
dw_filecount	On success returns the number of entries in the array of pointers to strings. The number returned is non-negative.
dw_error	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()

9.11.2.2 dwarf_srclines_b()

```
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

dw_cudie	The Compilation Unit (CU) DIE of interest.
dw_version_out	The DWARF Line Table version number (Standard: 2,3,4, or 5) Version 0xf006 is an experimental (two-level) line table.
dw_table_count	Zero or one means this is a normal DWARF line table. Two means this is an experimental two-level line table.
dw_linecontext	On success sets the pointer to point to an opaque structure usable for further queries.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.3 dwarf_srclines_from_linecontext()

```
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

dw_linecontext	The line context of interest.
dw_linebuf	On success returns an array of pointers to Dwarf_Line.
dw_linecount	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.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.4 dwarf_srclines_two_level_from_linecontext()

```
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.

Works for DWARF2,3,4,5 and for experimental two-level line tables. A single level table will have ∗linebuf_actuals and ∗linecount_actuals 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_actuals will be 0.

People not using these two-level tables should dwarf_srclines_from_linecontext instead.

9.11.2.5 dwarf_srclines_dealloc_b()

```
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

dw_context	The context to be dealloced (freed). On return the pointer passed in is stale and calling applications should zero the pointer.

9.11.2.6 dwarf_srclines_table_offset()

```
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

dw_context	
dw_offset	On success returns the section offset of the dw_context.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.7 dwarf_srclines_comp_dir()

```
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

dw_context	The Line Context of interest.
dw_compilation_directory	On success returns a pointer to a string identifying the compilation directory of the CU.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.8 dwarf_srclines_subprog_count()

```
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

dw_context	The Dwarf_Line_Context of interest.
dw_count	On success returns the two-level line table subprogram array size in this line context.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.9 dwarf_srclines_subprog_data()

```
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

dw_context	The Dwarf_Line_Context of interest.
dw_index	The item to retrieve. Valid indexes are 1 through dw_count.
dw_name	On success returns a pointer to the subprog name.
dw_decl_file	On success returns a file number through the pointer.
dw_decl_line	On success returns a line number through the pointer.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.10 dwarf_srclines_files_indexes()

```
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

dw_context	The line context of interest.
dw_baseindex	On success returns the base index of valid file indexes. With DWARF2,3,4 the value is 1. With DWARF5 the value is 0.
dw_count	On success returns the real count of entries.
dw_endindex	On success returns value such that callers should index as dw_baseindex through dw_endindex-1.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

See also

Using dwarf_srclines_b()

9.11.2.11 dwarf_srclines_files_data_b()

```
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

dw_context	The line context of interest.
dw_index_in	The entry of interest. Callers should index as dw_baseindex through dw_endindex-1.
dw_name	If dw_name non-null on success returns The file name in the line table header through the pointer.
dw_directory_index	If dw_directory_index non-null on success returns the directory number in the line table header through the pointer.
dw_last_mod_time	If dw_last_mod_time non-null on success returns the directory last modification date/time through the pointer.
dw_file_length	If dw_file_length non-null on success returns the file length recorded in the line table through the pointer.
dw_md5ptr	If dw_md5ptr 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.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.12 dwarf_srclines_include_dir_count()

```
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

dw_line_context	The line context of interest.
dw_count	On success returns the count of directories. How to use this depends on the line table version number.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

See also

dwarf_srclines_include_dir_data

9.11.2.13 dwarf_srclines_include_dir_data()

```
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

dw_line_context	The line context of interest.
dw_index	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.
dw_name	On success it returns a pointer to a directory name. Do not free/deallocate the string.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

See also

dwarf_srclines_include_dir_count

9.11.2.14 dwarf_srclines_version()

```
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

dw_line_context	The Line Context of interest.
dw_version	On success, returns the line table version through the pointer.
dw_table_count	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.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.15 dwarf_linebeginstatement()

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

Read Line beginstatement register.

Link to Line Table Registers

Parameters

dw_line	The Dwarf_Line of interest.
dw_returned_bool	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.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.16 dwarf_lineendsequence()

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

Read Line endsequence register flag.

Link to Line Table Registers

Parameters

dw_line	The Dwarf_Line of interest.
dw_returned_bool	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.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.17 dwarf_lineno()

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

Read Line line register.

Link to Line Table Registers

Parameters

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

Returns

DW_DLV_OK if it succeeds.

9.11.2.18 dwarf_line_srcfileno()

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

Read Line file register.

Link to Line Table Registers

Parameters

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

Returns

DW_DLV_OK if it succeeds.

9.11.2.19 dwarf_line_is_addr_set()

```
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

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

Returns

DW_DLV_OK if it succeeds.

9.11.2.20 dwarf_lineaddr()

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

Return the address of the Dwarf_Line.

Link to Line Table Registers

Parameters

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

Returns

DW_DLV_OK if it succeeds.

9.11.2.21 dwarf_lineoff_b()

```
int dwarf_lineoff_b (
            Dwarf_Line dw_line,
```

```
            Dwarf_Unsigned * dw_returned_lineoffset,
            Dwarf_Error * dw_error )
```

Return a column number through the pointer.

Link to Line Table Registers

Parameters

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

Returns

DW_DLV_OK if it succeeds.

9.11.2.22 dwarf_linesrc()

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

Return the file name applicable to the Dwarf_Line.

Link to Line Table Registers

Parameters

dw_line	The Dwarf_Line of interest.
dw_returned_name	On success it reads the file register and finds the source file name from the line table header and returns a pointer to that file name string through the pointer.
dw_error	The usual error pointer. Do not dealloc or free the string.

Returns

DW_DLV_OK if it succeeds.

9.11.2.23 dwarf_lineblock()

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

Return the basic_block line register.

Link to Line Table Registers

Parameters

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

Returns

DW_DLV_OK if it succeeds.

9.11.2.24 dwarf_prologue_end_etc()

```
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.

Link to Line Table Registers

Parameters

dw_line	The Dwarf_Line of interest.
dw_prologue_end	On success it sets the flag to TRUE or FALSE from the prologue_end register in the line table.
dw_epilogue_begin	On success it sets the flag to TRUE or FALSE from the epilogue_begin register in the line table.
dw_isa	On success it sets the value to the value of from the isa register in the line table.
dw_discriminator	On success it sets the value to the value of from the discriminator register in the line table.
dw_error	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.25 dwarf_check_lineheader_b()

```
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

Dwarf_Printf_Callback_Info_s

Parameters

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

Returns

DW_DLV_OK etc.

9.11.2.26 dwarf_print_lines()

```
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.

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

See also

dwarf_check_lineheader_b

Dwarf_Printf_Callback_Info_s

Parameters

dw_cu_die	The CU DIE of interest
dw_error	
dw_errorcount_out	

Returns

DW_DLV_OK etc.

9.11.2.27 dwarf_register_printf_callback()

```
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.

For the structure you must fill in:

See also

Dwarf_Printf_Callback_Info_s

Parameters

dw_dbg	The Dwarf_Debug of interest.
dw_callbackinfo	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.

9.12 Ranges: code addresses in DWARF3-4

Functions

- 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.
- void dwarf_dealloc_ranges (Dwarf_Debug dw_dbg, Dwarf_Ranges *dw_rangesbuf, Dwarf_Signed dw_↩ rangecount)

  Dealloc the array dw_rangesbuf.

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_get_ranges_b()

```
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

dw_dbg	The Dwarf_Debug of interest
dw_rangesoffset	The offset to read from in the section.
dw_die	Pass in the DIE whose DW_AT_ranges brought us to ranges.
dw_return_realoffset	The actual offset in the section actually read. In a tieddbg this
dw_rangesbuf	A pointer to an array of structs is returned here.
dw_rangecount	The count of structs in the array is returned here.
dw_bytecount	The number of bytes in the .debug_ranges section applying to the returned array. This makes possible just marching through the section by offset.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.12.2.2 dwarf_dealloc_ranges()

```
void dwarf_dealloc_ranges (
            Dwarf_Debug dw_dbg,
```

```
Dwarf_Ranges * dw_rangesbuf,
Dwarf_Signed dw_rangecount )
```

Dealloc the array dw_rangesbuf.

Parameters

dw_dbg	The Dwarf_Debug of interest.
dw_rangesbuf	The dw_rangesbuf pointer returned by dwarf_get_ranges_b
dw_rangecount	The dw_rangecount returned by dwarf_get_ranges_b

9.13 Rnglists: code addresses in DWARF5

Functions

- 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.
- 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.
- void dwarf_dealloc_rnglists_head (Dwarf_Rnglists_Head dw_head)

  Dealloc a Dwarf_Rnglists_Head.
- int dwarf_load_rnglists (Dwarf_Debug dw_dbg, Dwarf_Unsigned *dw_rnglists_count, Dwarf_Error *dw_↩ error)

  Loads all .debug_rnglists headers.
- 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.
- 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 rangelists.
- 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.
- 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_rnglists_get_rle_head()

```
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 rangelists .debug_rnglists DW_FORM_sec_offset DW↩ _FORM_rnglistx (DW_AT_ranges in DWARF5).

See also

Accessing a rnglists section

Parameters

dw_attr	The attribute referring to .debug_rnglists
dw_theform	The form number, DW_FORM_sec_offset or DW_FORM_rnglistx.
dw_index_or_offset_value	If the form is an index, pass it here. If the form is an offset, pass that here.
dw_head_out	On success creates a record owning the rnglists data for this attribute.
dw_count_of_entries_in_head	On success this is set to the number of entry in the rnglists for this attribute.
dw_global_offset_of_rle_set	On success set to the global offset of the rnglists in the rnglists section.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.13.2.2 dwarf_get_rnglists_entry_fields_a()

```
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 a rnglists section

Parameters

dw_head	The Dwarf_Rnglists_Head of interest.
dw_entrynum	Valid values are 0 through dw_count_of_entries_in_head-1.
dw_entrylen	On success returns the length in bytes of this individual entry.
dw_rle_value_out	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.
dw_raw1	On success returns a value directly recorded in the rangelist entry if that applies to this rle.
dw_raw2	On success returns a value directly recorded in the rangelist entry if that applies to this rle.
dw_debug_addr_unavailable	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.
dw_cooked1	On success returns (if appropriate) the dw_raw1 value turned into a valid address.
dw_cooked2	On success returns (if appropriate) the dw_raw2 value turned into a valid address. Ignore the value if dw_debug_addr_unavailable is set.
dw_error	The usual error detail return pointer. Ignore the value if dw_debug_addr_unavailable is set.

Returns

Returns DW_DLV_OK etc.

9.13.2.3 dwarf_dealloc_rnglists_head()

```
void dwarf_dealloc_rnglists_head (
            Dwarf_Rnglists_Head dw_head )
```

Dealloc a Dwarf_Rnglists_Head.

Parameters

dw_head	dealloc all the memory associated with dw_head. The caller should then immediately set the pointer to zero/NULL as it is stale.

9.13.2.4 dwarf_load_rnglists()

```
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

Accesing 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

dw_dbg	
dw_rnglists_count	On success it returns the number of rnglists headers in the section through dw_rnglists_count.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. If the section does not exist the function returns DW_DLV_OK.

9.13.2.5 dwarf_get_rnglist_offset_index_value()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_context_index	Begin this at zero.
dw_offsetentry_index	Begin this at zero.
dw_offset_value_out	On success returns the rangelist entry offset within the rangelist set.
dw_global_offset_value_out	On success returns the rangelist entry offset within rnglist section.
dw_error	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.

9.13.2.6 dwarf_get_rnglist_head_basics()

```
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 rangelists.

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.

9.13.2.7 dwarf_get_rnglist_context_basics()

```
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.

9.13.2.8 dwarf_get_rnglist_rle()

```
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.

9.14 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

- 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.
- 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.
- 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 of a location expression.
- 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.
- 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_Small 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.
- void dwarf_dealloc_loc_head_c (Dwarf_Loc_Head_c dw_head)

  Dealloc (free) all memory allocated for Dwarf_Loc_Head_c.
- int dwarf_load_loclists (Dwarf_Debug dw_dbg, Dwarf_Unsigned *dw_loclists_count, Dwarf_Error *dw_error)

  Load Loclists.
- 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.
- 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.
- 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.
- 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_get_loclist_c()

```
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

dw_attr	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..
dw_loclist_head	On success returns a pointer to the created loclist head record.
dw_locentry_count	On success returns the count of records. For an expression it will be one.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.14.2.2 dwarf_get_loclist_head_kind()

```
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

dw_loclist_head	Pass in a loclist head pointer.
dw_lkind	On success returns the loclist kind through the pointer. For example DW_LKIND_expression.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.14.2.3 dwarf_get_locdesc_entry_d()

```
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 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

dw_loclist_head	A loclist head pointer.
dw_index	Pass in an index value less than dw_locentry_count .
dw_lle_value_out	On success returns the DW_LLE value applicable, such as DW_LLE_start_end .
dw_rawlowpc	On success returns the first operand in the expression (if the expression has an operand).
dw_rawhipc	On success returns the second operand in the expression. (if the expression has a second operand).
dw_debug_addr_unavailable	On success returns FALSE if the data required to calculate dw_lowpc_cooked or dw_hipc_cooked was present or TRUE if some required data was missing (for example in split dwarf).
dw_lowpc_cooked	On success and if dw_debug_addr_unavailable FALSE returns the true low address.
dw_hipc_cooked	On success and if dw_debug_addr_unavailable FALSE returns the true high address.
dw_locexpr_op_count_out	On success returns the count of operations in the expression.
dw_locentry_out	On success returns a pointer to a specific location description.
dw_loclist_source_out	On success returns the applicable DW_LKIND value.
dw_expression_offset_out	On success returns the offset of the expression in the applicable section.
dw_locdesc_offset_out	On return sets the offset to the location description offset (if that is meaningful) or zero for simple location expressions.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.14.2.4 dwarf_get_location_op_value_c()

```
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

dw_locdesc	Pass in a valid Dwarf_Locdesc_c.
dw_index	Pass in the operator index. zero through dw_locexpr_op_count_out-1.
dw_operator_out	On success returns the DW_OP operator, such as DW_OP_plus .
dw_operand1	On success returns the value of the operand or zero.
dw_operand2	On success returns the value of the operand or zero.
dw_operand3	On success returns the value of the operand or zero.
dw_offset_for_branch	On success returns The byte offset of the operator within the entire expression. Useful for checking the correctness of operators that branch..
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.14.2.5 dwarf_loclist_from_expr_c()

```
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_Small 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

dw_dbg	The applicable Dwarf_Debug
dw_expression_in	Pass in a pointer to the expression bytes.
dw_expression_length	Pass in the length, in bytes, of the expression.
dw_address_size	Pass in the applicable address_size.
dw_offset_size	Pass in the applicable offset size.
dw_dwarf_version	Pass in the applicable dwarf version.
dw_loc_head	On success returns a pointer to a dwarf location head record for use in getting to the details of the expression.
dw_listlen	On success, sets the listlen to one.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.14.2.6 dwarf_dealloc_loc_head_c()

```
void dwarf_dealloc_loc_head_c (
            Dwarf_Loc_Head_c dw_head )
```

Dealloc (free) all memory allocated for Dwarf_Loc_Head_c.

Parameters

dw_head	A head pointer.

The caller should zero the passed-in pointer on return as it is stale at that point.

9.14.2.7 dwarf_load_loclists()

```
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

dw_dbg	The applicable Dwarf_Debug.
dw_loclists_count	On success, returns the number of DWARF5 loclists contexts in the section, whether this is the first or a duplicate load.
dw_error	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.

9.14.2.8 dwarf_get_loclist_offset_index_value()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_context_index	Pass in the loclists context index.
dw_offsetentry_index	Pass in the offset array index.
dw_offset_value_out	On success returns the offset value at offset table[dw_offsetentry_index], an offset local to this context.
dw_global_offset_value_out	On success returns the same offset value but with the offset of the table added in to form a section offset.
dw_error	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.

9.14.2.9 dwarf_get_loclist_head_basics()

```
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.

9.14.2.10 dwarf_get_loclist_context_basics()

```
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.

9.14.2.11 dwarf_get_loclist_lle()

```
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.

9.15 .debug_addr access: DWARF5

Functions

- 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.
- 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.
- 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_debug_addr_table()

```
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 .debug_addr section (by dwarfdump). Not at all clear it is of any other use.

Parameters

dw_dbg	The Dwarf_Debug of interest.
dw_section_offset	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.
dw_table_header	On success Returns a pointer to a Dwarf_Debug_Addr_Table for use with dwarf_get_attr_by_index().
dw_length	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.
dw_version	On success returns the version number, which should be 5.
dw_address_size	On success returns the address size of the address entries in this table.
dw_at_addr_base	On success returns the value that will appear in some DW_AT_addr_base attribute.
dw_entry_count	On success returns the number of table entries in this table instance.
dw_next_table_offset	On success returns the offset of the next table in the section. Use the offset returned in the next call to this function.
dw_error	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.

9.15.2.2 dwarf_debug_addr_by_index()

```
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

dw_dat	Pass in a Dwarf_Debug_Addr_Table pointer.
dw_entry_index	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.
dw_address	Returns an address in the program through the pointer.
dw_error	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_dealloc_debug_addr_table()

```
void dwarf_dealloc_debug_addr_table (
            Dwarf_Debug_Addr_Table dw_dat )
```

dealloc (free) a Dwarf_Attr_Table record.

Parameters

dw_dat	Pass in a valid Dwarf_Debug_Addr_Table pointer. Does nothing if the dw_dat field is NULL.

9.16 Macro Access: DWARF5

Functions

- 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.
- 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.
- 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.
- void dwarf_dealloc_macro_context (Dwarf_Macro_Context dw_mc)

Dealloc a macro context.

- 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.
- 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.
- 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.
- 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.
- 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.
- 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

examplep5 An example reading .debug_macro

9.16.2 Function Documentation

9.16.2.1 dwarf_get_macro_context()

```
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

examplep5

Parameters

dw_die	The CU DIE of interest.
dw_version_out	On success returns the macro context version (5)
dw_macro_context	On success returns a pointer to a macro context which allows access to the context content.
dw_macro_unit_offset_out	On success returns the offset of the macro context.
dw_macro_ops_count_out	On success returns the number of macro operations in the context.
dw_macro_ops_data_length_out	On success returns the length in bytes of the operations in the context.
dw_error	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.

9.16.2.2 dwarf_get_macro_context_by_offset()

```
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

dw_die	The CU DIE of interest.
dw_offset	The offset in the section to begin reading.
dw_version_out	On success returns the macro context version (5)
dw_macro_context	On success returns a pointer to a macro context which allows access to the context content.
dw_macro_ops_count_out	On success returns the number of macro operations in the context.
dw_macro_ops_data_length	On success returns the length in bytes of the macro context, starting at the offset of the first byte of the context.
dw_error	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.

9.16.2.3 dwarf_macro_context_total_length()

```
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

dw_context	A pointer to the macro context of interest.
dw_mac_total_len	On success returns the length in bytes of the macro context.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.16.2.4 dwarf_dealloc_macro_context()

```
void dwarf_dealloc_macro_context (
            Dwarf_Macro_Context dw_mc )
```

Dealloc a macro context.

Parameters

dw_mc	A pointer to the macro context of interest. On return the caller should zero the pointer as the pointer is then stale.

9.16.2.5 dwarf_macro_context_head()

```
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.

9.16.2.6 dwarf_macro_operands_table()

```
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

dw_mc	The macro context of interest.
dw_index	The opcode operands table index. 0 through dw_opcode_count-1.
dw_opcode_number	On success returns the opcode number in the table.
dw_operand_count	On success returns the number of forms for that dw_index.
dw_operand_array	On success returns the array of op operand forms
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.16.2.7 dwarf_get_macro_op()

```
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

dw_macro_context	The macro context of interest.
dw_op_number	valid values are 0 through dw_macro_ops_count_out-1.

Parameters

dw_op_start_section_offset	On success returns the section offset of this operator.
dw_macro_operator	On success returns the the macro operator itself, for example DW_MACRO_define.
dw_forms_count	On success returns the number of forms in the formcode array.
dw_formcode_array	On success returns a pointer to the formcode array of operand forms.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.16.2.8 dwarf_get_macro_defundef()

```
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

dw_macro_context	The macro context of interest.
dw_op_number	valid values are 0 through dw_macro_ops_count_out-1. The op number must be for a def/undef.
dw_line_number	The line number in the user source for this define/undef
dw_index	On success if the macro is an strx form the value returned is the string index in the record, otherwise zero is returned.
dw_offset	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.
dw_forms_count	On success the value 2 is returned.
dw_macro_string	On success a pointer to a null-terminated string is returned. Do not dealloc or free this string.
dw_error	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_sup, DW_MACRO_define_strx, or DW_MACRO_undef_strx,

9.16.2.9 dwarf_get_macro_startend_file()

```
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

dw_macro_context	The macro context of interest.
dw_op_number	Valid values are 0 through dw_macro_ops_count_out-1. The op number must be for a start/end.
dw_line_number	If end_file nothing is returned here. If start_file on success returns the line number of the source line of the include directive.
dw_name_index_to_line_tab	If end_file nothing is returned here. If start_file on success returns the file name index in the line table file names table.
dw_src_file_name	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.
dw_error	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.

9.16.2.10 dwarf_get_macro_import()

```
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

dw_macro_context	The macro context of interest.
dw_op_number	Valid values are 0 through dw_macro_ops_count_out-1.
dw_target_offset	Returns the offset in the imported section.
dw_error	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.

9.17 Macro Access: DWARF2-4

Functions

- char * dwarf_find_macro_value_start (char *dw_macro_string)

  Return a pointer to the value part of a macro.
- 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()

```
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

dw_macro_string	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()

```
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.

An example calling this function

See also

Reading .debug_macinfo (DWARF2-4)

Parameters

dw_dbg	The Dwarf_Debug of interest.
dw_macro_offset	The offset in the section you wish to start from.
dw_maximum_count	Pass in a count to ensure we will not allocate an excessive amount (guarding against a
dw_entry_count	On success returns a count of the macro operations in a CU macro set.
dw_details	On success returns a pointer to an array of struct DW_Macro_Details_s .
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18 Stack Frame Access

Functions

- 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.
- 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.
- 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.

- 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.
- 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.
- int dwarf_get_cie_of_fde (Dwarf_Fde dw_fde, Dwarf_Cie *dw_cie_returned, Dwarf_Error *dw_error)

  Given FDE get CIE.
- 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.
- int dwarf_get_cie_index (Dwarf_Cie dw_cie, Dwarf_Signed *dw_index, Dwarf_Error *dw_error)

  Return CIE index given CIE.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.

- 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.
- 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.
- 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.
- 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.
- 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.
- 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)
- void dwarf_dealloc_frame_instr_head (Dwarf_Frame_Instr_Head dw_head)

  Deallocates the frame instruction data in dw_head.
- 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.
- 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.
- 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 .
- Dwarf_Half dwarf_set_frame_rule_initial_value (Dwarf_Debug dw_dbg, Dwarf_Half dw_value)

  Frame Rule Initial Value.
- Dwarf_Half dwarf_set_frame_cfa_value (Dwarf_Debug dw_dbg, Dwarf_Half dw_value)

  Frame CFA Column Invariants for setting frame registers .
- Dwarf_Half dwarf_set_frame_same_value (Dwarf_Debug dw_dbg, Dwarf_Half dw_value)

  Frame Same Value Default Invariants for setting frame registers .
- 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 Function Documentation

9.18.2.1 dwarf_get_fde_list()

```
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 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

dw_dbg	The Dwarf_Debug of interest.
dw_cie_data	On success returns a pointer to an array of pointers to CIE data.
dw_cie_element_count	On success returns a count of the number of elements in the dw_cie_data array.
dw_fde_data	On success returns a pointer to an array of pointers to FDE data.
dw_fde_element_count	On success returns a count of the number of elements in the dw_fde_data array. On success
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.2.2 dwarf_get_fde_list_eh()

```
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

9.18.2.3 dwarf_dealloc_fde_cie_list()

```
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

dw_dbg	The Dwarf_Debug used in the list setup.
dw_cie_data	As returned from the list setup call.
dw_cie_element_count	
dw_fde_data	As returned from the list setup call.
dw_fde_element_count	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.

9.18.2.4 dwarf_get_fde_range()

```
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

dw_fde	The FDE of interest.
dw_low_pc	On success returns the low pc value for the function involved.
dw_func_length	On success returns the length of the function code in bytes.
dw_fde_bytes	On success returns a pointer to the bytes of the FDE.
dw_fde_byte_length	On success returns the length of the dw_fde_bytes area.
dw_cie_offset	On success returns the section offset of the associated CIE.
dw_cie_index	On success returns the CIE index of the associated CIE.
dw_fde_offset	On success returns the section offset of this FDE.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.2.5 dwarf_get_fde_exception_info()

```
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.

9.18.2.6 dwarf_get_cie_of_fde()

```
int dwarf_get_cie_of_fde (
            Dwarf_Fde dw_fde,
            Dwarf_Cie * dw_cie_returned,
            Dwarf_Error * dw_error )
```

Given FDE get CIE.

Parameters

dw_fde	The FDE of interest.
dw_cie_returned	On success returns a pointer to the applicable CIE.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.2.7 dwarf_get_cie_info_b()

```
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

dw_cie	Pass in the CIE of interest.
dw_bytes_in_cie	On success, returns the length of the CIE in bytes.
dw_version	On success, returns the CIE version number.
dw_augmenter	On success, returns a pointer to the augmentation string (which could be the empty string).
dw_code_alignment_factor	On success, returns a the code_alignment_factor used to interpret CIE/FDE operations.
dw_data_alignment_factor	On success, returns a the data_alignment_factor used to interpret CIE/FDE operations.
dw_return_address_register_rule	On success, returns a register number of the return address register.
dw_initial_instructions	On success, returns a pointer to the bytes of initial_instructions in the CIE.
dw_initial_instructions_length	On success, returns the length in bytes of the initial_instructions.
dw_offset_size	On success, returns the offset_size within this CIE.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.2.8 dwarf_get_cie_index()

```
int dwarf_get_cie_index (
            Dwarf_Cie dw_cie,
            Dwarf_Signed * dw_index,
            Dwarf_Error * dw_error )
```

Return CIE index given CIE.

Parameters

dw_cie	Pass in the CIE of interest.
dw_index	On success, returns the index (the position of the CIE in the CIE pointer array).
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.2.9 dwarf_get_fde_instr_bytes()

```
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

dw_fde	Pass in the FDE of interest.
dw_outinstrs	On success returns a pointer to the FDE instruction byte stream.
dw_outlen	On success returns the length of the dw_outinstrs byte stream.
dw_error	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.2.10 dwarf_get_fde_info_for_all_regs3_b()

```
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.

Parameters

dw_fde	Pass in the FDE of interest.
dw_pc_requested	Pass in a pc (code) address inside that FDE.
dw_reg_table	On success, returns a pointer to a struct given the frame state.
dw_row_pc	On success returns the address of the row of frame data which may be a few counts off of the pc requested.
dw_has_more_rows	On success returns FALSE if there are no more rows, otherwise returns TRUE.
dw_subsequent_pc	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.
dw_error	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.

9.18.2.11 dwarf_get_fde_info_for_all_regs3()

```
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.

If you need to iterate through all rows of the FDE, consider switching to dwarf_get_fde_info_for_all_regs3_b() as it is more efficient.

9.18.2.12 dwarf_get_fde_info_for_reg3_c()

```
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 one could instead call dwarf_get_fde_info_for_all_regs3() and index into the table it fills in.

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.

Parameters

dw_fde	Pass in the FDE of interest.
dw_table_column	Pass in the table_column, column numbers in the table are 0 through the number_of_registers-1.
dw_pc_requested	Pass in the pc of interest within dw_fde.
dw_value_type	On success returns the value type, a DW_EXPR value. For example DW_EXPR_EXPRESSION
dw_offset_relevant	On success returns FALSE if the offset value is irrelevant, otherwise TRUE.
dw_register	On success returns a register number.
dw_offset	On success returns a signed register offset value when dw_value_type is DW_EXPR_OFFSET or DW_EXPER_VAL_OFFSET.
dw_block_content	On success returns a pointer to a block. For example, for DW_EXPR_EXPRESSION the block gives access to the expression bytes.
dw_row_pc_out	On success returns the address of the actual pc for this register at this pc.
dw_has_more_rows	On success returns FALSE if there are no more rows, otherwise returns TRUE.
dw_subsequent_pc	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.
dw_error	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.

9.18.2.13 dwarf_get_fde_info_for_reg3_b()

```
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()

9.18.2.14 dwarf_get_fde_info_for_cfa_reg3_c()

```
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).

9.18.2.15 dwarf_get_fde_info_for_cfa_reg3_b()

```
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.

9.18.2.16 dwarf_get_fde_for_die()

```
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.

9.18.2.17 dwarf_get_fde_n()

```
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

9.18.2.18 dwarf_get_fde_at_pc()

```
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

dw_fde_data	Pass in a pointer an array of fde pointers.
dw_pc_of_interest	The pc value of interest.
dw_returned_fde	On success a pointer to the applicable FDE is set through the pointer.
dw_lopc	On success a pointer to the low pc in dw_returned_fde is set through the pointer.
dw_hipc	On success a pointer to the high pc (one past the actual last byte address) in dw_returned_fde is set through the pointer.
dw_error	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.

9.18.2.19 dwarf_get_cie_augmentation_data()

```
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

dw_cie	The CIE of interest.
dw_augdata	On success returns a pointer to the augmentation data.
dw_augdata_len	On success returns the length in bytes of the augmentation data.
dw_error	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.

9.18.2.20 dwarf_get_fde_augmentation_data()

```
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

dw_fde	The FDE of interest.
dw_augdata	On success returns a pointer to the augmentation data.
dw_augdata_len	On success returns the length in bytes of the augmentation data.
dw_error	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.

9.18.2.21 dwarf_expand_frame_instructions()

```
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

dw_cie	The cie relevant to the instructions.
dw_instructionspointer	points to the instructions
dw_length_in_bytes	byte length of the instruction sequence.
dw_head	The address of an allocated dw_head
dw_instr_count	Returns the number of instructions in the byte stream
dw_error	Error return details

Returns

On success returns DW_DLV_OK

9.18.2.22 dwarf_get_frame_instruction()

```
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

dw_head	A head record
dw_instr_index	index 0 < i < instr_count
dw_instr_offset_in_instrs	Returns the byte offset of this instruction within instructions.
dw_cfa_operation	Returns a DW_CFA opcode.
dw_fields_description	Returns a string. Do not free.
dw_u0	May be set to an unsigned value
dw_u1	May be set to an unsigned value
dw_s0	May be set to a signed value
dw_s1	May be set to a signed value
dw_code_alignment_factor	May be set by the call
dw_data_alignment_factor	May be set by the call
dw_expression_block	Pass in a pointer to a block
dw_error	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.

9.18.2.23 dwarf_get_frame_instruction_a()

```
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.

9.18.2.24 dwarf_dealloc_frame_instr_head()

```
void dwarf_dealloc_frame_instr_head (
            Dwarf_Frame_Instr_Head dw_head )
```

Deallocates the frame instruction data in dw_head.

Parameters

dw_head	A head pointer. Frees all data created by dwarf_expand_frame_instructions() and makes the head pointer stale. The caller should set to NULL.

9.18.2.25 dwarf_fde_section_offset()

```
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

dw_dbg	The Dwarf_Debug of interest
dw_in_fde	Pass in the FDE of interest.
dw_fde_off	On success returns the section offset of the FDE.
dw_cie_off	On success returns the section offset of the CIE.
dw_error	Error return details

Returns

Returns DW_DLV_OK etc.

9.18.2.26 dwarf_cie_section_offset()

```
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

dw_dbg	The Dwarf_Debug of interest
dw_in_cie	Pass in the CIE of interest.
dw_cie_off	On success returns the section offset of the CIE.
dw_error	Error return details

Returns

Returns DW_DLV_OK etc.

9.18.2.27 dwarf_set_frame_rule_table_size()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_value	Pass in the value to record for the library to use.

Returns

Returns the previous value.

9.18.2.28 dwarf_set_frame_rule_initial_value()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_value	Pass in the value to record for the library to use.

Returns

Returns the previous value.

9.18.2.29 dwarf_set_frame_cfa_value()

```
Dwarf_Half dwarf_set_frame_cfa_value (
            Dwarf_Debug dw_dbg,
            Dwarf_Half dw_value )
```

Frame CFA Column Invariants for setting frame registers .

Parameters

dw_dbg	The Dwarf_Debug of interest.
dw_value	Pass in the value to record for the library to use.

Returns

Returns the previous value.

9.18.2.30 dwarf_set_frame_same_value()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_value	Pass in the value to record for the library to use.

Returns

Returns the previous value.

9.18.2.31 dwarf_set_frame_undefined_value()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_value	Pass in the value to record for the library to use.

Returns

Returns the previous value.

9.19 Abbreviations Section Details

Functions

- 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.
- int dwarf_get_abbrev_tag (Dwarf_Abbrev dw_abbrev, Dwarf_Half *dw_return_tag_number, Dwarf_Error *dw_error)

  Get abbreviation tag.
- int dwarf_get_abbrev_code (Dwarf_Abbrev dw_abbrev, Dwarf_Unsigned *dw_return_code_number, Dwarf_Error *dw_error)

  Get Abbreviation Code.
- int dwarf_get_abbrev_children_flag (Dwarf_Abbrev dw_abbrev, Dwarf_Signed *dw_return_flag, Dwarf_Error *dw_error)

  Get Abbrev Children Flag.
- 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.

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()

```
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_d

Parameters

dw_dbg	The Dwarf_Debug of interest.
dw_offset	Pass in the offset where a Debug_Abbrev starts.
dw_returned_abbrev	On success, sets a pointer to a Dwarf_Abbrev through the pointer to allow further access.
dw_length	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.
dw_attr_count	On success, returns the number of attributes in this abbreviation entry.
dw_error	On error dw_error 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_tag()

```
int dwarf_get_abbrev_tag (
            Dwarf_Abbrev dw_abbrev,
            Dwarf_Half * dw_return_tag_number,
            Dwarf_Error * dw_error )
```

Get abbreviation tag.

Parameters

dw_abbrev	The Dwarf_Abbrev of interest.
dw_return_tag_number	Returns the tag value, for example DW_TAG_compile_unit.
dw_error	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()

```
int dwarf_get_abbrev_code (
            Dwarf_Abbrev dw_abbrev,
            Dwarf_Unsigned * dw_return_code_number,
            Dwarf_Error * dw_error )
```

Get Abbreviation Code.

Parameters

dw_abbrev	The Dwarf_Abbrev of interest.
dw_return_code_number	Returns the code for this abbreviation, a number assigned to the abbreviation and unique within the applicable CU.
dw_error	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_children_flag()

```
int dwarf_get_abbrev_children_flag (
            Dwarf_Abbrev dw_abbrev,
```

```
            Dwarf_Signed * dw_return_flag,
            Dwarf_Error * dw_error )
```

Get Abbrev Children Flag.

Parameters

dw_abbrev	The Dwarf_Abbrev of interest.
dw_return_flag	On success returns the flag TRUE (greater than zero) if the DIE referencing the abbreviation has children, else returns FALSE (zero).
dw_error	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.19.2.5 dwarf_get_abbrev_entry_b()

```
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 will call with filter_outliers non-zero.

Parameters

dw_abbrev	The Dwarf_Abbrev of interest.
dw_indx	Valid dw_index values are 0 through dw_attr_count-1
dw_filter_outliers	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)).
dw_returned_attr_num	On success returns the attribute number, such as DW_AT_name
dw_returned_form	On success returns the attribute FORM, such as DW_FORM_udata
dw_returned_implicit_const	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..
dw_offset	On success returns the offset of the start of this attr/form pair in the abbreviation section.
dw_error	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.

9.20 String Section .debug_str Details

Functions

- 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()

```
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

dw_dbg	The Dwarf_Debug whose .debug_str section we want to access.
dw_offset	Pass in a a string offset. Start at 0, and for the next call pass in dw_offset plus dw_strlen_of_string plus 1.
dw_string	On success returns a pointer to a string from offset dw_offset. Never dealloc or free this string.
dw_strlen_of_string	On success returns the strlen() of the string.
dw_error	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 or if dw_offset is >= the section size it returns DW_DLV_NO_ENTRY.

9.21 Str_Offsets section details

Functions

- 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.
- 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.
- 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.
- 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.
- 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_open_str_offsets_table_access()

```
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

dw_dbg	Pass in the Dwarf_Debug of interest.
dw_table_data	On success returns a pointer to an opaque structure for use in further calls.
dw_error	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

9.21.2.2 dwarf_close_str_offsets_table_access()

```
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

dw_table_data	
dw_error	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 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)..

9.21.2.3 dwarf_next_str_offsets_table()

```
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

dw_table_data	Pass in an open Dwarf_Str_Offsets_Table.
dw_unit_length	On success returns a table unit_length field
dw_unit_length_offset	On success returns the section offset of the unit_length field.
dw_table_start_offset	On success returns the section offset of the array of table entries.
dw_entry_size	On success returns the entry size (4 or 8)
dw_version	On success returns the value in the version field 5.
dw_padding	On success returns the zero value in the padding field.
dw_table_value_count	On success returns the number of table entries, each of size dw_entry_size, in the table.
dw_error	On error dw_error is set to point to the error details.

Returns

DW_DLV_OK Returns DW_DLV_NO_ENTRY if there are no more entries.

9.21.2.4 dwarf_str_offsets_value_by_index()

```
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

dw_table_data	Pass in the open table pointer.
dw_index_to_entry	Pass in the entry number, 0 through dw_table_value_count-1 for the active table
dw_entry_value	On success returns the value in that table entry, an offset into a string table.
dw_error	On error dw_error is set to point to the error details.

Returns

DW_DLV_OK Returns DW_DLV_ERROR if dw_index_to_entry is out of the correct range.

9.21.2.5 dwarf_str_offsets_statistics()

```
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

dw_table_data	Pass in the open table pointer.
dw_wasted_byte_count	Always returns 0 at present.
dw_table_count	On success returns the total number of tables seen so far in the section.
dw_error	On error dw_error is set to point to the error details.

Returns

DW_DLV_OK etc.

9.22 Dwarf_Error Functions

Functions

- Dwarf_Unsigned dwarf_errno (Dwarf_Error dw_error)

  What DW_DLE code does the error have?
- char * dwarf_errmsg (Dwarf_Error dw_error)

  What message string is in the error?
- char * dwarf_errmsg_by_number (Dwarf_Unsigned dw_errornum)

  What message string is associated with the error number.
- 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.
- 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_errno()

```
Dwarf_Unsigned dwarf_errno (
            Dwarf_Error dw_error )
```

What DW_DLE code does the error have?

Parameters

dw_error	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.

9.22.2.2 dwarf_errmsg()

```
char* dwarf_errmsg (
            Dwarf_Error dw_error )
```

What message string is in the error?

Parameters

dw_error	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()

```
char* dwarf_errmsg_by_number (
            Dwarf_Unsigned dw_errornum )
```

What message string is associated with the error number.

Parameters

dw_errornum	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_error_creation()

```
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

dw_dbg	The relevant Dwarf_Debug.
dw_error	a Dwarf_Error is returned through this pointer.
dw_errmsg	The message string you provide.

9.22.2.5 dwarf_dealloc_error()

```
void dwarf_dealloc_error (
            Dwarf_Debug dw_dbg,
            Dwarf_Error dw_error )
```

Free (dealloc) an Dwarf_Error something created.

Parameters

dw_dbg	The relevant Dwarf_Debug pointer.
dw_error	A pointer to a Dwarf_Error. The pointer is then stale so you should immediately zero that pointer passed in.

9.23 Generic dwarf_dealloc Function

Functions

- 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.

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()

```
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

dw_dbg	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.
dw_space	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.
dw_type	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

- 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()

```
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

- 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.
- void dwarf_dealloc_dnames (Dwarf_Dnames_Head dw_dn)

  Frees all the malloc data associated with dw_dn.
- 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.
- 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.

- 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.
- 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.
- 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.
- 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.
- 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.
- 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_dnames_header()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_starting_offset	Read this section starting at offset zero.
dw_dn	On success returns a pointer to a set of data allowing access to the table.
dw_offset_of_next_table	On success returns Offset just past the end of the the opened table.
dw_error	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.

9.25.2.2 dwarf_dealloc_dnames()

```
void dwarf_dealloc_dnames (
            Dwarf_Dnames_Head dw_dn )
```

Frees all the malloc data associated with dw_dn.

Parameters

dw_dn	A Dwarf_Dnames_Head pointer. Callers should zero the pointer passed in as soon as possible after this returns as the pointer is then stale.

9.25.2.3 dwarf_dnames_abbrevtable()

```
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

dw_dn	A Dwarf_Dnames_Head pointer.
dw_index	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.

Parameters

dw_abbrev_offset	Returns the offset of the abbrev table entry for this names table entry.
dw_abbrev_code	Returns the abbrev code for the abbrev at offset dw_abbrev_offset.
dw_abbrev_tag	Returns the tag for the abbrev at offset dw_abbrev_offset.
dw_array_size	The size you allocated in each of the following two arrays.
dw_idxattr_array	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.
dw_form_array	Pass in an array you allocated where the function returns and array of forms for this dw_abbrev_code (paralled to dw_idxattr_array). The last form code in the array is zero.
dw_idxattr_count	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.

9.25.2.4 dwarf_dnames_sizes()

```
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).

9.25.2.5 dwarf_dnames_offsets()

```
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"

9.25.2.6 dwarf_dnames_cu_table()

```
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

dw_dn	The table of interest.
dw_type	Pass in the type, "cu" or "tu"
dw_index_number	For "cu" index range is 0 through K-1 For "tu" index range is 0 through T+F-1
dw_offset	Zero if it cannot be determined. (check the return value!).
dw_sig	the Dwarf_Sig8 is filled in with a signature if the TU index is T through T+F-1
dw_error	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.25.2.7 dwarf_dnames_bucket()

```
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

dw_dn	The Dwarf_Dnames_Head of interest.
dw_bucket_number	Pass in a bucket number Bucket numbers start at 0.
dw_index	On success returns the index of the appropriate name entry. Name entry indexes start at one, a zero index means the bucket is unused.
dw_indexcount	On success returns the number of name entries in the bucket.
dw_error	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

9.25.2.8 dwarf_dnames_name()

```
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

dw_dn	The table of interest.
dw_name_index	Pass in the desired index, start at one.
dw_bucket_number	On success returns a bucket number, zero if no buckets present.
dw_hash_value	The hash value, all zeros if no hashes present
dw_offset_to_debug_str	The offset to the .debug_str section string.
dw_ptrtostr	if dw_ptrtostr non-null returns a pointer to the applicable string here.
dw_offset_in_entrypool	Returns the offset in the entrypool
dw_abbrev_number	Returned from entrypool.
dw_abbrev_tag	Returned from entrypool abbrev data.
dw_array_size	Size of array you provide to hold DW_IDX index attribute and form numbers. Possibly 10 suffices for practical purposes.
dw_idxattr_array	Array space you provide, for idx attribute numbers (function will initialize it). The final entry in the array will be 0.
dw_form_array	Array you provide, for form numbers (function will initialize it). The final entry in the array will be 0.
dw_idxattr_count	Array entries needed. Might be larger than dw_array_size, meaning not all entries could be returned in your array.
dw_error	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_entrypool()

```
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.

Returns the basic data about an entrypool record and enables correct calling of dwarf_dnames_entrypool_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_entrypool_values()

Parameters

dw_dn	Pass in the debug names table of interest.
dw_offset_in_entrypool	The record offset (in the entry pool table) of the first record of IDX attributes. Starts at zero.
dw_abbrev_code	On success returns the abbrev code of the idx attributes for the pool entry.

Parameters

dw_tag	On success returns the TAG of the DIE referred to by this entrypool entry.
dw_value_count	On success returns the number of distinct values imply by this entry.
dw_index_of_abbrev	On success returns the index of the abbrev index/form pairs in the abbreviation table.
dw_offset_of_initial_value	On success returns the entry pool offset of the sequence of bytes containing values, such as a CU index or a DIE offset.
dw_error	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.

9.25.2.10 dwarf_dnames_entrypool_values()

```
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

dw_dn	Pass in the debug names table of interest.
dw_index_of_abbrev	Pass in the abbreviation index.
dw_offset_in_entrypool_of_values	Pass in the offset of the values returned by dw_offset_of_initial_value above.
dw_arrays_length	Pass in the array length of each of the following four fields. The dw_value_count returned above is what you need to use.
dw_array_idx_number	Create an array of Dwarf_Half values, dw_arrays_length long, and pass a pointer to the first entry here.

Parameters

dw_array_form	Create an array of Dwarf_Half values, dw_arrays_length long, and pass a pointer to the first entry here.
dw_array_of_offsets	Create an array of Dwarf_Unsigned values, dw_arrays_length long, and pass a pointer to the first entry here.
dw_array_of_signatures	Create an array of Dwarf_Sig8 structs, dw_arrays_length long, and pass a pointer to the first entry here.
dw_offset_of_next_entrypool	On success returns the offset of the next entrypool. A value here is usable in the next call to dwarf_dnames_entrypool.
dw_single_cu	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.
dw_cu_offset	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.
dw_error	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.

9.26 Fast Access to a CU given a code address

Functions

- 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.
- 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.
- 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.
- 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.
- 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_aranges()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_aranges	On success returns a pointer to an array of Dwarf_Arange pointers.
dw_arange_count	On success returns a count of the length of the array.
dw_error	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.

9.26.2.2 dwarf_get_arange()

```
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

dw_aranges	Pass in a pointer to the first entry in the aranges array of pointers.
dw_arange_count	Pass in the dw_arange_count, the count for the array.
dw_address	Pass in the code address of interest.
dw_returned_arange	On success, returns the particular arange that holds that address.
dw_error	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 code address present in the section.

9.26.2.3 dwarf_get_cu_die_offset()

```
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

dw_arange	The specific arange of interest.
dw_return_offset	The CU DIE offset (in .debug_info) applicable to this arange..
dw_error	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_arange_cu_header_offset()

```
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

dw_arange	The specific arange of interest.
dw_return_cu_header_offset	The CU header offset (in .debug_info) applicable to this arange.
dw_error	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.26.2.5 dwarf_get_arange_info_b()

```
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

dw_arange	The specific arange of interest.
dw_segment	On success and if segment_entry_size is non-zero this returns the segment number from the arange.
dw_segment_entry_size	On success returns the segment entry size from the arange.
dw_start	On success returns the low address this arange refers to.
dw_length	On success returns the length, in bytes of the code area this arange refers to.
dw_cu_die_offset	On success returns the .debug_info section offset the arange refers to.
dw_error	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_typenames */
- #define **DW_GL_VARS** 4 /* .debug_varnames */
- #define **DW_GL_WEAKS** 5 /* .debug_weaknames */

Functions

- 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.
- int **dwarf_get_pubtypes** (Dwarf_Debug dw_dbg, Dwarf_Global **dw_pubtypes, Dwarf_Signed *dw_↩ number_of_pubtypes, Dwarf_Error *dw_error)
- 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.
- void dwarf_globals_dealloc (Dwarf_Debug dw_dbg, Dwarf_Global *dw_global_like, Dwarf_Signed dw_count)

Dealloc the Dwarf_Global data.

- int dwarf_globname (Dwarf_Global dw_global, char **dw_returned_name, Dwarf_Error *dw_error)

  Return the name of a global-like data item.
- 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.
- 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.
- 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.
- Dwarf_Half dwarf_global_tag_number (Dwarf_Global dw_global)

  Return the DW_TAG number of a global entry.
- 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.
- 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()

```
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.

The code here, as of 0.4.3, September 3 2022, returns data from either section.

See also

Using dwarf_get_globals()

Parameters

dw_dbg	The Dwarf_Debug of interest.
dw_globals	On success returns an array of pointers to opaque structs..
dw_number_of_globals	On success returns the number of entries in the array.
dw_error	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_globals_by_type()

```
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. Simplfies access by replace dwarf_get_pubtypes, dwarf_get_funcs, dwarf_get_types, dwarfget_vars, and dwarf_get_weaks with a single set of types.

Parameters

dw_dbg	The Dwarf_Debug of interest.
dw_requested_section	Pass in one of the values DW_GL_GLOBALS through DW_GL_WEAKS to select the section to extract data from.
dw_contents	On success returns an array of pointers to opaque structs.
dw_count	On success returns the number of entries in the array.
dw_error	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.

9.27.2.3 dwarf_globals_dealloc()

```
void dwarf_globals_dealloc (
            Dwarf_Debug dw_dbg,
            Dwarf_Global * dw_global_like,
            Dwarf_Signed dw_count )
```

Dealloc the Dwarf_Global data.

Parameters

dw_dbg	The Dwarf_Debug of interest.
dw_global_like	The array of globals/types/etc data to dealloc (free).
dw_count	The number of entries in the array.

9.27.2.4 dwarf_globname()

```
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

dw_global	The Dwarf_Global of interest.
dw_returned_name	On success a pointer to the name (a null-terminated string) is returned.
dw_error	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()

```
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

dw_global	The Dwarf_Global of interest.
dw_die_offset	On success a the section-global DIE offset of a data item is returned.
dw_error	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_cu_offset()

```
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

dw_global	The Dwarf_Global of interest.
dw_cu_header_offset	On success a the section-global offset of a CU header is returned.
dw_error	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_name_offsets()

```
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

dw_global	The Dwarf_Global of interest.
dw_returned_name	On success a pointer to the name (a null-terminated string) is returned.
dw_die_offset	On success a the section-global DIE offset of the global with the name.
dw_cu_die_offset	On success a the section-global offset of the relevant CU DIE is returned.
dw_error	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.27.2.8 dwarf_global_tag_number()

```
Dwarf_Half dwarf_global_tag_number (
            Dwarf_Global dw_global )
```

Return the DW_TAG number of a global entry.

Parameters

dw_global	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.

9.27.2.9 dwarf_get_globals_header()

```
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.

9.27.2.10 dwarf_return_empty_pubnames()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_flag	Must be the value one.

Returns

Returns DW_DLV_OK. Always.

9.28 Fast Access to GNU .debug_gnu_pubnames

Functions

- 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.
- void dwarf_gnu_index_dealloc (Dwarf_Gnu_Index_Head dw_head)

  Free resources of .debug_gnu_pubnames .debug_gnu_pubtypes.
- 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.
- 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_head()

```
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

dw_dbg	Pass in the Dwarf_Debug of interest.
dw_which_section	Pass in TRUE to access .debug_gnu_pubnames. Pass in FALSE to access .debug_gnu_typenames.
dw_head	On success, set to a pointer to a head record allowing access to all the content of the section.
dw_index_block_count_out	On success, set to a count of the number of blocks of data available.
dw_error	

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_gnu_index_dealloc()

```
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

dw_head	Pass in the Dwarf_Gnu_Index_head whose data is to be deallocated.

9.28.2.3 dwarf_get_gnu_index_block()

```
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

dw_head	Pass in the Dwarf_Gnu_Index_head interest.
dw_number	Pass in the block number of the block of interest. 0 through dw_index_block_count_out-1.

Parameters

dw_block_length	On success set to the length of the data in this block, in bytes.
dw_version	On success set to the version number of the block.
dw_offset_into_debug_info	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, sucn a possibility does not cause the function to return DW_DLV_ERROR.
dw_size_of_debug_info_area	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_are plus dw_offset_into_debug_info may be outside the bounds of the actual .debug_info section, sucn 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.
dw_count_of_index_entries	On success set to the count of index entries in this particlular block number.
dw_error	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.

9.28.2.4 dwarf_get_gnu_index_block_entry()

```
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

dw_head	Pass in the Dwarf_Gnu_Index_head interest.
dw_blocknumber	Pass in the block number of the block of interest. 0 through dw_index_block_count_out-1.
dw_entrynumber	Pass in the entry number of the entry of interest. 0 through dw_count_of_index_entries-1.
dw_offset_in_debug_info	On success set to the offset in .debug_info relevant to this entry.
dw_name_string	On success set to the size in bytes, in .debug_info, of the area this block refersto.
dw_flagbyte	On success set to the entry flag byte content.
dw_staticorglobal	On success set to the entry static/global letter.
dw_typeofentry	On success set to the type of entry.
dw_error	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.

9.29 Fast Access to Gdb Index

Functions

- 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.
- void dwarf_dealloc_gdbindex (Dwarf_Gdbindex dw_gdbindexptr)

  Free (dealloc) all allocated Dwarf_Gdbindex memory It should named dwarf_dealloc_gdbindex.
- int dwarf_gdbindex_culist_array (Dwarf_Gdbindex dw_gdbindexptr, Dwarf_Unsigned *dw_list_length, Dwarf_Error *dw_error)

  Return the culist array length.
- 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.
- 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.
- 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.
- int dwarf_gdbindex_addressarea (Dwarf_Gdbindex dw_gdbindexptr, Dwarf_Unsigned *dw_addressarea_↩ list_length, Dwarf_Error *dw_error)

  Get access to gdbindex address area.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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_gdbindex_header()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_gdbindexptr	On success returns a pointer to make access to table details possible.
dw_version	On success returns the table version.
dw_cu_list_offset	On success returns the offset of the cu_list in the section.
dw_types_cu_list_offset	On success returns the offset of the types cu_list in the section.
dw_address_area_offset	On success returns the area pool offset.
dw_symbol_table_offset	On success returns the symbol table offset.
dw_constant_pool_offset	On success returns the constant pool offset.

Returns

Returns DW_DLV_OK etc. Returns DW_DLV_NO_ENTRY if the section is absent.

9.29.2.2 dwarf_dealloc_gdbindex()

```
void dwarf_dealloc_gdbindex (
            Dwarf_Gdbindex dw_gdbindexptr )
```

Free (dealloc) all allocated Dwarf_Gdbindex memory It should named dwarf_dealloc_gdbindex.

Parameters

dw_gdbindexptr	Pass in a valid dw_gdbindexptr and on return assign zero to dw_gdbindexptr as it is stale.

9.29.2.3 dwarf_gdbindex_culist_array()

```
int dwarf_gdbindex_culist_array (
            Dwarf_Gdbindex dw_gdbindexptr,
            Dwarf_Unsigned * dw_list_length,
            Dwarf_Error * dw_error )
```

Return the culist array length.

Parameters

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_list_length	On success returns the array length of the cu list.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.4 dwarf_gdbindex_culist_entry()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_entryindex	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.
dw_cu_offset	On success returns the CU offet for this list entry.
dw_cu_length	On success returns the CU length(in bytes) for this list entry.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.5 dwarf_gdbindex_types_culist_array()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_types_list_length	On success returns the array length of the types cu list.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.6 dwarf_gdbindex_types_culist_entry()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_types_entryindex	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.
dw_cu_offset	On success returns the types CU offet for this list entry.
dw_tu_offset	On success returns the tu offet for this list entry.
dw_type_signature	On success returns the type unit offset for this entry if the type has a signature.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.7 dwarf_gdbindex_addressarea()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_addressarea_list_length	On success returns the number of entries in the addressarea.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.8 dwarf_gdbindex_addressarea_entry()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_entryindex	Pass in an index, 0 through dw_addressarea_list_length-1. addressarea.
dw_low_address	On success returns the low address for the entry.
dw_high_address	On success returns the high address for the entry.
dw_cu_index	On success returns the index to the cu for the entry.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.9 dwarf_gdbindex_symboltable_array()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_symtab_list_length	On success returns the number of entries in the symbol table
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.10 dwarf_gdbindex_symboltable_entry()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_entryindex	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;
dw_string_offset	On success returns the string offset in the appropriate string section.
dw_cu_vector_offset	On success returns the CU vector offset.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.11 dwarf_gdbindex_cuvector_length()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_cuvector_offset	Pass in the offset, dw_cu_vector_offset.
dw_innercount	On success returns the number of CUs in the cuvector instance array.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.12 dwarf_gdbindex_cuvector_inner_attributes()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_cuvector_offset_in	Pass in the value of dw_cuvector_offset
dw_innerindex	Pass in the index of the CU vector in, from 0 through dw_innercount-1.
dw_field_value	On success returns a field of bits. To expand the bits call dwarf_gdbindex_cuvector_instance_expand_value.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.13 dwarf_gdbindex_cuvector_instance_expand_value()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_field_value	Pass in the dw_field_value returned by dwarf_gdbindex_cuvector_inner_attributes.
dw_cu_index	On success returns the CU index from the dw_field_value
dw_symbol_kind	On success returns the symbol kind (see the sourceware page. Kinds are TYPE, VARIABLE, or FUNCTION.
dw_is_static	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.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.14 dwarf_gdbindex_string_by_offset()

```
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

dw_gdbindexptr	Pass in the Dwarf_Gdbindex pointer of interest.
dw_stringoffset	Pass in the string offset returned by dwarf_gdbindex_symboltable_entry
dw_string_ptr	On success returns a a pointer to the null-terminated string.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.30 Fast Access to Split Dwarf (Debug Fission)

Functions

- 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.
- void dwarf_dealloc_xu_header (Dwarf_Xu_Index_Header dw_xuhdr)

  Dealloc (free) memory associated with dw_xuhdr.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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_get_xu_index_header()

```
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

dw_dbg	Pass in the Dwarf_Debug of interest
dw_section_type	Pass in a pointer to either "cu" or "tu".
dw_xuhdr	On success, returns a pointer usable in further calls.
dw_version_number	On success returns five.
dw_section_count	On success returns the number of entries in the table of section counts. Referred to as **N**.
dw_units_count	On success returns the number of compilation units or type units in the index. Referred to as **U**.
dw_hash_slots_count	On success returns the number of slots in the hash table. Referred to as **S**.
dw_sect_name	On success returns a pointer to the name of the section. Do not free/dealloc the returned pointer.
dw_error	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.

9.30.2.2 dwarf_dealloc_xu_header()

```
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

dw_xuhdr	Dealloc (free) all associated memory. The caller should zero the passed in value on return as it is then a stale value.

9.30.2.3 dwarf_get_xu_index_section_type()

```
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

dw_xuhdr	Pass in an open header pointer.
dw_typename	On success returns a pointer to the immutable string "tu" or "cu". Do not free.
dw_sectionname	On success returns a pointer to the section name in the object file. Do not free.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.30.2.4 dwarf_get_xu_hash_entry()

```
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

dw_xuhdr	Pass in an open header pointer.
dw_index	Pass in the index of the entry you wish. Valid index values are 0 through **S-1**. If the dw_index passed in is outside the valid range the functionj
dw_hash_value	Pass in a pointer to a Dwarf_Sig8. On success the hash struct is filled in with the 8 byte hash value.
dw_index_to_sections	On success returns the offset/size table index for this hash entry.
dw_error	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_section_names()

```
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

dw_xuhdr	Pass in an open header pointer.
dw_column_index	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 dw_column_index values are 0 through **N-1**.
dw_SECT_number	On success returns DW_SECT_INFO or other section id as appears in dw_column_index.
dw_SECT_name	On success returns a pointer to the string for with the section name.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.30.2.6 dwarf_get_xu_section_offset()

```
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.

Parameters

dw_xuhdr	Pass in an open header pointer.
dw_row_index	Pass in a row number , 1 through **U**
dw_column_index	Pass in a column number , 0 through **N-1**
dw_sec_offset	On success returns the section offset of the section whose name dwarf_get_xu_section_names returns.
dw_sec_size	On success returns the section size of the section whose name dwarf_get_xu_section_names returns.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.30.2.7 dwarf_get_debugfission_for_die()

```
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

dw_die	Pass in a Dwarf_Die pointer, Usually pass in a CU DIE pointer.
dw_percu_out	Pass in a pointer to a zeroed structure. On success the function fills in the structure.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.30.2.8 dwarf_get_debugfission_for_key()

```
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

dw_dbg	Pass in the Dwarf_Debug of interest.
dw_hash_sig	Pass in a pointer to a Dwarf_Sig8 containing a hash value of interest.
dw_cu_type	Pass in the type, a string. Either "cu" or "tu".
dw_percu_out	Pass in a pointer to a zeroed structure. On success the function fills in the structure.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.31 Access GNU .gnu_debuglink, build-id.

Functions

- 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.
- int dwarf_suppress_debuglink_crc (int dw_suppress)

  Suppressing crc calculations.
- int dwarf_add_debuglink_global_path (Dwarf_Debug dw_dbg, const char *dw_pathname, Dwarf_Error *dw_error)

  Adding debuglink global paths.
- int dwarf_crc32 (Dwarf_Debug dw_dbg, unsigned char *dw_crcbuf, Dwarf_Error *dw_error)

  Crc32 used for debuglink crc calculation.
- 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 is separate from a normal shared object. Has nothing to do with split-dwarf/debug-fission.

9.31.2 Function Documentation

9.31.2.1 dwarf_gnu_debuglink()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_debuglink_path_returned	On success returns a pointer to a path in the debuglink section. Do not free!
dw_crc_returned	On success returns a pointer to a 4 byte area through the pointer.
dw_debuglink_fullpath_returned	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.
dw_debuglink_path_length_returned	On success returns the strlen() of dw_debuglink_fullpath_returned .
dw_buildid_type_returned	On success returns a pointer to integer with a type code. See the buildid definition.
dw_buildid_owner_name_returned	On success returns a pointer to the owner name from the buildid section. Do not free this.

Parameters

dw_buildid_returned	On success returns a pointer to a sequence of bytes containing the buildid.
dw_buildid_length_returned	On success this is set to the length of the set of bytes pointed to by dw_buildid_returned .
dw_paths_returned	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).
dw_paths_length_returned	On success returns the length of the array of string pointers dw_paths_returned points at.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.31.2.2 dwarf_suppress_debuglink_crc()

```
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

dw_suppress	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 dwarf_init_path_dl calls compute debuglink CRC values as required.

Returns

Returns the previous value of the global flag.

Details on separate DWARF object access

9.31.2.3 dwarf_add_debuglink_global_path()

```
int dwarf_add_debuglink_global_path (
            Dwarf_Debug dw_dbg,
            const char * dw_pathname,
            Dwarf_Error * dw_error )
```

Adding debuglink global paths.

Only really inside 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

dw_dbg	Pass in the Dwarf_Debug of interest.
dw_pathname	Pass in a pathname to add to the list of global paths used by debuglink.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.31.2.4 dwarf_crc32()

```
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 is filled in.

Parameters

dw_dbg	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.
dw_crcbuf	Pass in a pointer to a 4 byte area to hold the returned crc, on success the function puts the 4 bytes there.
dw_error	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.31.2.5 dwarf_basic_crc32()

```
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.

Parameters

dw_buf	Pass in a pointer to some bytes on which the crc calculation as done in debuglink is to be done.
dw_len	Pass in the length in bytes of dw_buf.
dw_init	Pass in the initial 32 bit value, zero is the right choice.

Returns

Returns an int (assumed 32 bits int!) with the calculated crc.

9.32 Harmless Error recording

Macros

- #define DW_HARMLESS_ERROR_CIRCULAR_LIST_DEFAULT_SIZE 4

  Default size of the libdwarf-internal circular list.

Functions

- 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.
- 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.
- 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 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.

9.32.2 Function Documentation

9.32.2.1 dwarf_get_harmless_error_list()

```
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 the 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

dw_dbg	The applicable Dwarf_Debug.
dw_count	The number of string buffers. If count is passed as zero no elements of the array are touched.
dw_errmsg_ptrs_array	A pointer to a user-created array of pointer to const char.
dw_newerr_count	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_set_harmless_error_list_size()

```
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

dw_dbg	The applicable Dwarf_Debug.
dw_maxcount	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.

9.32.2.3 dwarf_insert_harmless_error()

```
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

dw_dbg	Pass in an open Dwarf_Debug
dw_newerror	Pass in a string whose content the function inserts as a harmless error (which dwarf_get_harmless_error_list will retrieve.

9.33 Names DW_TAG_member etc as strings

Functions

- int dwarf_get_ACCESS_name (unsigned int dw_val_in, const char **dw_s_out)
  dwarf_get_ACCESS_name
- int dwarf_get_ADDR_name (unsigned int dw_val_in, const char **dw_s_out)
  dwarf_get_ADDR_name
- int dwarf_get_AT_name (unsigned int dw_val_in, const char **dw_s_out)
  dwarf_get_AT_name
- int dwarf_get_ATCF_name (unsigned int dw_val_in, const char **dw_s_out)
  dwarf_get_AT_name
- int dwarf_get_ATE_name (unsigned int dw_val_in, const char **dw_s_out)
  dwarf_get_ATE_name
- int dwarf_get_CC_name (unsigned int dw_val_in, const char **dw_s_out)

dwarf_get_CC_name

- int dwarf_get_CFA_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_CFA_name
- int dwarf_get_children_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_children_namea - historic misspelling.
- int dwarf_get_CHILDREN_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_CHILDREN_name
- int dwarf_get_DEFAULTED_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_DEFAULTED_name
- int dwarf_get_DS_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_DS_name
- int dwarf_get_DSC_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_DSC_name
- int dwarf_get_GNUIKIND_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_GNUIKIND_name - libdwarf invention
- int dwarf_get_EH_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_EH_name
- int dwarf_get_END_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_END_name
- int dwarf_get_FORM_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_FORM_name
- int dwarf_get_FRAME_name (unsigned int dw_val_in, const char **dw_s_out)

  This is a set of register names.
- int dwarf_get_GNUIVIS_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_GNUIVIS_name - a libdwarf invention
- int dwarf_get_ID_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_ID_name
- int dwarf_get_IDX_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_IDX_name
- int dwarf_get_INL_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_INL_name
- int dwarf_get_ISA_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_ISA_name
- int dwarf_get_LANG_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_LANG_name
- int dwarf_get_LLE_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_LLE_name
- int dwarf_get_LLEX_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_LLEX_name - a GNU extension.
- int dwarf_get_LNCT_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_LNCT_name
- int dwarf_get_LNE_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_LNE_name
- int dwarf_get_LNS_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_LNS_name
- int dwarf_get_MACINFO_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_MACINFO_name
- int dwarf_get_MACRO_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_MACRO_name
- int dwarf_get_OP_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_OP_name

- int dwarf_get_ORD_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_ORD_name
- int dwarf_get_RLE_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_RLE_name
- int dwarf_get_SECT_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_SECT_name
- int dwarf_get_TAG_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_TAG_name
- int dwarf_get_UT_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_UT_name
- int dwarf_get_VIRTUALITY_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_VIRTUALITY_name
- int dwarf_get_VIS_name (unsigned int dw_val_in, const char **dw_s_out)

  dwarf_get_VIS_name
- 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 are in static storage and must not be freed.

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.

See also

Retrieving tag,attribute,etc names

9.33.2 Function Documentation

9.33.2.1 dwarf_get_GNUIKIND_name()

```
int dwarf_get_GNUIKIND_name (
            unsigned int dw_val_in,
            const char ** dw_s_out )
```

dwarf_get_GNUIKIND_name - libdwarf invention

So we can report things GNU extensions sensibly.

9.33.2.2 dwarf_get_EH_name()

```
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.

9.33.2.3 dwarf_get_FRAME_name()

```
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_GNUIVIS_name()

```
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.

9.33.2.5 dwarf_get_LLEX_name()

```
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.

9.33.2.6 dwarf_get_MACINFO_name()

```
int dwarf_get_MACINFO_name (
            unsigned int dw_val_in,
            const char ** dw_s_out )
```

dwarf_get_MACINFO_name

Used in DWARF2-DWARF4

9.33.2.7 dwarf_get_MACRO_name()

```
int dwarf_get_MACRO_name (
            unsigned int dw_val_in,
            const char ** dw_s_out )
```

dwarf_get_MACRO_name

Used in DWARF5

9.33.2.8 dwarf_get_FORM_CLASS_name()

```
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.

9.34 Object Sections Data

Functions

- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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 from Dwarf_Debug.

- int dwarf_get_section_count (Dwarf_Debug dw_dbg)

  Get section count (of object file sections).

- 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_die_section_name()

```
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

dw_dbg	The Dwarf_Debug of interest
dw_is_info	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.
dw_sec_name	On success returns a pointer to the actual section name in the object file. Do not free the string.
dw_error	The usual error argument to report error details.

Returns

DW_DLV_OK etc.

9.34.2.2 dwarf_get_die_section_name_b()

```
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.

9.34.2.3 dwarf_get_real_section_name()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_std_section_name	Pass in a standard section name, such as .debug_info or .debug_info.dwo .
dw_actual_sec_name_out	On success returns the actual section name from the object file.
dw_marked_zcompressed	On success returns TRUE if the original section name ends in .zdebug
dw_marked_zlib_compressed	On success returns TRUE if the section has the ZLIB string at the front of the section.
dw_marked_shf_compressed	On success returns TRUE if the section flag (Elf SHF_COMPRESSED) is marked as compressed.
dw_compressed_length	On success if the section was compressed it returns the original section length in the object file.
dw_uncompressed_length	On success if the section was compressed this returns the uncompressed length of the object section.
dw_error	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

9.34.2.4 dwarf_get_frame_section_name()

```
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()

```
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_offset_size()

```
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.

9.34.2.7 dwarf_get_address_size()

```
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.

9.34.2.8 dwarf_get_line_section_name_from_die()

```
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

dw_die	Pass in a Dwarf_Die pointer.
dw_section_name_out	On success returns the section name, usually some .debug_info∗ name but in DWARF4 could be a .debug_types∗ name.
dw_error	On error returns the usual error pointer.

Returns

Returns DW_DLV_OK etc.

9.34.2.9 dwarf_get_section_info_by_name_a()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_section_name	Pass in a pointer to a section name. It must be an exact match to the real section name.
dw_section_addr	On success returns the section address as defined by an object header.
dw_section_size	On success returns the section size as defined by an object header.
dw_section_flags	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**.
dw_section_offset	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.
dw_error	On error returns the usual error pointer.

Returns

Returns DW_DLV_OK etc.

9.34.2.10 dwarf_get_section_info_by_name()

```
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.

9.34.2.11 dwarf_get_section_info_by_index_a()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_section_index	Pass in an index, 0 through N-1 where N is the count returned from dwarf_get_section_count .
dw_section_name	On success returns a pointer to the section name as it appears in the object file.
dw_section_addr	On success returns the section address as defined by an object header.
dw_section_size	On success returns the section size as defined by an object header.
dw_section_flags	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.
dw_section_offset	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.
dw_error	On error returns the usual error pointer.

Returns

Returns DW_DLV_OK etc.

9.34.2.12 dwarf_get_section_info_by_index()

```
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.

9.34.2.13 dwarf_machine_architecture()

```
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 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.

dwarf_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

dw_dbg	The Dwarf_Debug of interest.
dw_ftype	Pass in a pointer. On success the value pointed to will be set to the the applicable DW_FTYPE value (see libdwarf.h).
dw_obj_pointersize	Pass in a pointer. On success the value pointed to will be set to the the applicable pointer size, which is almost always either 4 or 8.
dw_obj_is_big_endian	Pass in a pointer. On success the value pointed to will be set to either 1 (the object being read is big-endia) or 0 (the object being read is little-endian.
dw_obj_machine	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.

Parameters

dw_obj_flags	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**).
dw_path_source	Pass in a pointer. On success the value pointed to will be set to a value that libdwarf sets to a DW_PATHSOURCE value indicating what caused the file path.
dw_ub_offset	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)
dw_ub_count	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)
dw_ub_index	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)
dw_comdat_groupnumber	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.

9.34.2.14 dwarf_get_section_max_offsets_d()

```
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

dw_dbg	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.

9.35 Section Groups Objectfile Data

Functions

- 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.
- 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 usually seen in relocatable Elf object files.

Section Groups Overview

9.35.2 Function Documentation

9.35.2.1 dwarf_sec_group_sizes()

```
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

dw_dbg	Pass in the Dwarf_Debug of interest.
dw_section_count_out	On success returns the number of DWARF sections in the object file. Can sometimes be many more than are of interest.
dw_group_count_out	On success returns the number of groups. Though usually one, it can be much larger.
dw_selected_group_out	On success returns the groupnumber that applies to this specific open Dwarf_Debug.
dw_map_entry_count_out	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.
dw_error	The usual error details pointer.

Returns

On success returns DW_DLV_OK

9.35.2.2 dwarf_sec_group_map()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_map_entry_count	Pass in the dw_map_entry_count_out from dwarf_sec_group_sizes
dw_group_numbers_array	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.
dw_sec_numbers_array	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.
dw_sec_names_array	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.
dw_error	The usual error details pointer.

Returns

On success returns DW_DLV_OK

9.36 LEB Encode and Decode

Functions

- int **dwarf_encode_leb128** (Dwarf_Unsigned dw_val, int *dw_nbytes, char *dw_space, int dw_splen)
- int **dwarf_encode_signed_leb128** (Dwarf_Signed dw_val, int *dw_nbytes, char *dw_space, int dw_splen)
- int **dwarf_decode_leb128** (char *dw_leb, Dwarf_Unsigned *dw_leblen, Dwarf_Unsigned *dw_outval, char *dw_endptr)
- 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

- const char * dwarf_package_version (void)

  Return the version string in the library.
- int dwarf_set_stringcheck (int dw_stringcheck)

  Turn off libdwarf checks of strings.
- int dwarf_set_reloc_application (int dw_apply)

  *Set libdwarf response to *.rela relocations.*
- 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.
- 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.
- Dwarf_Small dwarf_set_default_address_size (Dwarf_Debug dw_dbg, Dwarf_Small dw_value)

  Set the address size on a Dwarf_Debug.
- int dwarf_get_universalbinary_count (Dwarf_Debug dw_dbg, Dwarf_Unsigned *dw_current_index, Dwarf_Unsigned *dw_available_count)

  Retrieve universal binary index.

Variables

- void(*)(void *, const void *, unsigned long) dwarf_get_endian_copy_function (Dwarf_Debug dw_dbg)

  Get a pointer to the applicable swap/noswap function.
- Dwarf_Cmdline_Options **dwarf_cmdline_options**

9.37.1 Detailed Description

9.37.2 Function Documentation

9.37.2.1 dwarf_package_version()

```
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

9.37.2.2 dwarf_set_stringcheck()

```
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

dw_stringcheck	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.

9.37.2.3 dwarf_set_reloc_application()

```
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

dw_apply	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.

9.37.2.4 dwarf_record_cmdline_options()

```
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

dw_dd_options	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()

```
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

dw_v	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()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_value	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_get_universalbinary_count()

```
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

dw_dbg	The Dwarf_Debug of interest.
dw_current_index	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).
dw_available_count	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.

9.37.3 Variable Documentation

9.37.3.1 dwarf_get_endian_copy_function

```
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

dw_dbg	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

- 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)

- 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)
- 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_APPLEUNIVERSAL DW_FTYPE_MACH_O DW_FTYPE_ELF DW_FTYPE_PE

9.39 Using dwarf_init_path()

A libdwarf initialization call.

A libdwarf initialization call.

An example calling dwarf_init_path() and dwarf_finish()

Parameters

path	Path to an object we wish to open.

groupnumber	*/ void 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;
	}
	if (res == DW_DLV_NO_ENTRY) {
	/* Nothing we can do */
	return;
	}
	printf("The file we actually opened is %s\n",
	true_pathbuf);
	/* Call libdwarf functions here */
	dwarf_finish(dbg);
	}

9.40 Using dwarf_init_path_dl()

Initialization focused on GNU debuglink data.

Initialization 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 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()

```
*/
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/libdwarfdd/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) {
        /*  Necessary call even though dbg is null!
            This avoids a memory leak.  */
        dwarf_dealloc_error(dbg,*error);
        *error = 0;
        return DW_DLV_NO_ENTRY;
    }
    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 DW_DLV_OK;
}
```

9.41 Using dwarf_attrlist()

Showing dwarf_attrlist()

Showing dwarf_attrlist()

```
*/
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;
}
```

9.42 Attaching a tied dbg

Attaching a tied dbg.

Attaching a tied dbg.

By convention, open the base 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 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..

Parameters

tieddbg	
error	

Returns

Returns whatever DW_DLV appropriate to the caller to deal with.

```
*/
int example2(Dwarf_Debug dbg, Dwarf_Debug tieddbg,
    Dwarf_Error *error)
{
    int res = 0;
    /*  The caller should have opened dbg
        on the debug shared 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(dbg,tieddbg,error);
    /*  Let your caller (who initialized the dbg
        values) deal with doing dwarf_finish()
    */
    return res;
}
```

9.43 Detaching a tied dbg

Detaching a tied dbg.

Detaching a tied dbg.

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..

```
*/
int example3(Dwarf_Debug dbg,Dwarf_Error *error)
{
    int res = 0;
    res = dwarf_set_tied_dbg(dbg,NULL,error);
    if (res != DW_DLV_OK) {
        /* Something went wrong*/
        return res;
    }
    return res;
}
```

9.44 Examining Section Group data

Accessing Section Group data.

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);
}
```

9.45 Using dwarf_siblingofb()

Accessing a DIE sibling.

Accessing a DIE sibling.

Access to each DIE on a sibling list

```
*/
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;
}
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;
}
```

9.46 Using dwarf_child()

Accessing a DIE child.

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;
    }
}
```

9.47 using dwarf_validate_die_sibling

A DIE tree validation.

A DIE tree validation.

Here we show how one uses dwarf_validate_die_sibling(). Dwarfdump uses this function as a part of its valdation 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_siblinof_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;
}
```

9.48 Example walking CUs(e)

Accessing all CUs looking for specific items(e).

Accessing all 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 throught 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,
    Dwarf_Die d);
int  my_needed_data_exists(struct myrecords_struct *myrecords);
/*  Loop on DIE tree. */
static void
record_die_and_siblingse(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_siblingse(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.
            No call to siblingof() is appropriate here.
            */
        record_die_and_siblingse(dbg,cu_die,is_info,
            0, myrec,error);
        dwarf_dealloc_die(cu_die);
    }
    /*  Found what we looked for */
    return DW_DLV_OK;
}
```

9.49 Example walking CUs(d)

Accessing all CUs looking for specific items(d).

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 throught 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;
}
```

9.50 Using dwarf_offdie_b()

Accessing a DIE by its offset.

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;
}
```

9.51 Using dwarf_offset_given_die()

Finding the section offset of a CU DIE and the DIE.

Finding the section offset of a CU DIE and the 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;
}
```

9.52 Using dwarf_attrlist()

Calling dwarf_attrlist()

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;
}
```

9.53 Using dwarf_offset_list()

Using dwarf_offset_list.

Using dwarf_offset_list.

An example calling dwarf_offset_list

Parameters

dbg	the Dwarf_Debug of interest
dieoffset	The section offset of a Dwarf_Die
is_info	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.
error	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;
}
```

9.54 Documenting Form_Block

Documents Form_Block content.

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
```

9.55 Using dwarf_discr_list()

Using dwarf_discr_list and dwarf_formblock.

Using dwarf_discr_list and 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

dw_dbg	The applicable Dwarf_Debug
dw_die	The applicable Dwarf_Die
dw_attr	The applicable Dwarf_Attribute
dw_attrnum,The	attribute number passed in to shorten this example a bit.
dw_isunsigned,The	attribute number passed in to shorten this example a bit.
dw_theform,The	form number passed in to shorten this example a bit.
dw_error	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_Dsc_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;
}
```

9.56 Location/expression access

Access to DWARF2-5 loclists and loc-expressions.

Access to 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;
}
```

9.57 Reading a location expression

Getting the details of a location expression.

Getting the details of a location expression.

See also

Location/expression access

```
*/
int example_locexprc(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;
}
```

9.58 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

path	Path to an object we wish to open.

Parameters

groupnumber	*/ int examplec(Dwarf_Die cu_die,Dwarf_Error *error)
	{
	/* EXAMPLE: DWARF2-DWARF5 access. */
	Dwarf_Line *linebuf = 0;
	Dwarf_Signed linecount = 0;
	Dwarf_Line *linebuf_actuals = 0;
	Dwarf_Signed linecount_actuals = 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. */
	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

Parameters

9.59 Using dwarf_srclines_b() and linecontext

Another example of using dwarf_srclines_b()

Another example of using dwarf_srclines_b()

See also

dwarf_srclines_b

dwarf_srclines_from_linecontext

dwarf_srclines_dealloc_b

```
*/
int exampled(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) {
        /* use line */
    }
    dwarf_srclines_dealloc_b(context);
    return DW_DLV_OK;
}
```

9.60 Using dwarf_srcfiles()

Getting source file names given a DIE.

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;
}
```

9.61 Using dwarf_get_globals()

Access to global symbol names.

Access to 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 .debug_names (if it 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;
}
```

9.62 Using dwarf_globals_by_type()

Reading .debug_pubtypes.

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;
}
```

9.63 Reading .debug_weaknames (nonstandard)

The section was IRIX/MIPS only.

The section was IRIX/MIPS only.

This section is an SGI/MIPS extension, not created by modern compilers. You

```
*/
int exampleh(Dwarf_Debug dbg,Dwarf_Error *error)
{
    Dwarf_Signed  count = 0;
    Dwarf_Global *weaks = 0;
    Dwarf_Signed  i = 0;
    int           res = 0;
    res = dwarf_globals_by_type(dbg,DW_GL_WEAKS,
        &weaks,&count,error);
    if (res != DW_DLV_OK) {
        return res;
    }
    for (i = 0; i < count; ++i) {
        /* use weaks[i] */
    }
    dwarf_globals_dealloc(dbg, weaks, count);
    return DW_DLV_OK;
}
```

9.64 Reading .debug_funcnames (nonstandard)

The section was IRIX/MIPS only.

The section 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 *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;
}
```

9.65 Reading .debug_types (nonstandard)

The section was IRIX/MIPS only.

The section was IRIX/MIPS only.

This section is an SGI/MIPS extension, not created by modern compilers.

```
*/
int examplel(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;
}
```

9.66 Reading .debug_varnames data (nonstandard)

The section was IRIX/MIPS only.

The section was IRIX/MIPS only.

This section is an SGI/MIPS extension, not created by modern compilers.

```
*/
int examplen(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;
}
#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 meaninful 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;
}
int            has_unchecked_import_in_list(void);
Dwarf_Unsigned get_next_import_from_list(void);
void           mark_this_offset_as_examined(
                  Dwarf_Unsigned macro_unit_offset);
void           add_offset_to_list(Dwarf_Unsigned 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;
}
/*
```

9.67 Reading .debug_macinfo (DWARF2-4)

Reading .debug_macinfo, DWARF2-4.

Reading .debug_macinfo, DWARF2-4.

```
 */
void functionusingsigned(Dwarf_Signed 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;
}
```

9.68 Extracting fde, cie lists.

Opening FDE and CIE lists.

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;
}
```

9.69 Reading the .eh_frame section

Access to .eh_frame.

Access to .eh_frame.

```
*/
int exampler(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;
    }
    return fres;
}
```

9.70 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;
}
```

9.71 Reading string offsets section data

examplestrngoffsets

examplestrngoffsets

An example accessing the string offsets section

Parameters

dbg	The Dwarf_Debug of interest.
dw_error	On error dw_error 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;
}
/*
```

9.72 Reading an aranges section

Reading .debug_aranges.

Reading .debug_aranges.

An example accessing the .debug_aranges section. Looking all the aranges entries. This example is not searching for anything.

Parameters

dbg	The Dwarf_Debug of interest.
dw_error	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;
}
```

9.73 Example getting .debug_ranges data

Accessing ranges data.

Accessing ranges data.

```
*/
void functionusingrange(Dwarf_Ranges *r);
int examplev(Dwarf_Debug dbg,Dwarf_Off rangesoffset,
    Dwarf_Die die, Dwarf_Error*error)
{
    Dwarf_Signed count = 0;
    Dwarf_Off  realoffset = 0;
    Dwarf_Ranges *rangesbuf = 0;
    Dwarf_Unsigned bytecount = 0;
    int res = 0;
    res = dwarf_get_ranges_b(dbg,rangesoffset,die,
        &realoffset,
        &rangesbuf,&count,&bytecount,error);
    if (res != DW_DLV_OK) {
        return res;
    }
    {
        Dwarf_Signed i = 0;
        for ( i = 0; i < count; ++i ) {
            Dwarf_Ranges *cur = rangesbuf+i;
            /* Use cur. */
            functionusingrange(cur);
        }
        dwarf_dealloc_ranges(dbg,rangesbuf,count);
    }
    return DW_DLV_OK;
}
```

9.74 Reading gdbindex data

Accessing gdbindex section data.

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;
}
```

9.75 Reading gdbindex addressarea

Accessing gdbindex addressarea data.

Accessing gdbindex addressarea data.

```
*/
int examplewgdbindex(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;
}
```

9.76 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 resl = 0;
        resl = dwarf_gdbindex_symboltable_entry(gdbindex,i,
            &symnameoffset,&cuvecoffset,
            error);
        if (resl != DW_DLV_OK) {
            return resl;
        }
        resl = dwarf_gdbindex_string_by_offset(gdbindex,
            symnameoffset,&name,error);
        if (resl != DW_DLV_OK) {
            return resl;
        }
        resl = dwarf_gdbindex_cuvector_length(gdbindex,
            cuvecoffset,&cuvec_len,error);
        if (resl != DW_DLV_OK) {
            return resl;
        }
        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;
}
```

9.77 Reading cu and tu Debug Fission data

Example using dwarf_get_xu_index_header.

Example using dwarf_get_xu_index_header.

```
*/
int exampley(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;
}
```

9.78 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;
}
```

9.79 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;
}
```

9.80 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;
}
```

9.81 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 = 0;
    int res = 0;
    /* The following is wrong, do not do it! */
    res = dwarf_get_ACCESS_name(DW_TAG_entry_point,&out);
    /*  Nothing one does here with 'res' or 'out'
        is meaningful. */
    /* The following is meaningful.*/
    res = dwarf_get_TAG_name(DW_TAG_entry_point,&out);
    if ( res == DW_DLV_OK) {
        /*  Here 'out' is a pointer one can use which
            points to the string "DW_TAG_entry_point". */
    } else {
        /*  Here 'out' has not been touched, it is
            uninitialized.  Do not use it. */
    }
}
```

9.82 Using GNU debuglink data

exampledebuglink Showing dwarf_add_debuglink_global_path

exampledebuglink Showing dwarf_add_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",builddowner_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 < paths_count; ++i ){
        printf("%2u: %s\n",i,paths[i]);
    }
    free(debuglink_fullpath);
    free(paths);
    return DW_DLV_OK;
}
```

9.83 Accesing accessing raw rnglist

Showing access to rnglist.

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;
}
```

9.84 Accessing a rnglists section

Showing access to rnglists on an Attribute.

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;
}
```

9.85 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 demonstates 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, };
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,
    gptrsize,
    gfilesize,
    gseccount,
    gloadsec,
    0 /* no relocating anything */
    };
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;
}
```

9.86 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 detals "
        "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

Data Structure Documentation

10.1 Dwarf_Block_s Struct Reference

Data Fields

- 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

Data Fields

- Dwarf_Bool **check_verbose_mode**

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

Data Fields

- const char * **pcu_type**
- Dwarf_Unsigned **pcu_index**
- Dwarf_Sig8 **pcu_hash**
- Dwarf_Unsigned **pcu_offset** [12]
- Dwarf_Unsigned **pcu_size** [12]
- 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

Data Fields

- 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>
```

Data Fields

- 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

Data Fields

- 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>`

Data Fields

- 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)

10.7.1 Detailed Description

The functions we need to access object data from libdwarf are declared here.

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

Data Fields

- 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**

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

Data Fields

- 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**

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

Data Fields

- 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

Data Fields

- 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

Data Fields

- 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

Data Fields

- 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

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 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
```

11.1 /home/davea/dwarf/code/src/bin/dwarfexample/jitreader.c File Reference

11.2 /home/davea/dwarf/code/src/bin/dwarfexample/showsectiongroups.c File Reference

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_" . libdwarf.h contains all the type declarations and function 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_ .

Index

.debug_addr access: DWARF5, 134
    dwarf_dealloc_debug_addr_table, 136
    dwarf_debug_addr_by_index, 135
    dwarf_debug_addr_table, 134
/home/davea/dwarf/code/src/bin/dwarfexample/jitreader.c, 291
/home/davea/dwarf/code/src/bin/dwarfexample/showsectiongroups.c, 291

A simple report on section groups., 281
Abbreviations Section Details, 163
    dwarf_get_abbrev, 164
    dwarf_get_abbrev_children_flag, 165
    dwarf_get_abbrev_code, 165
    dwarf_get_abbrev_entry_b, 166
    dwarf_get_abbrev_tag, 165
Accesing accessing raw rnglist, 274
Access GNU .gnu_debuglink, build-id., 211
    dwarf_add_debuglink_global_path, 213
    dwarf_basic_crc32, 215
    dwarf_crc32, 214
    dwarf_gnu_debuglink, 211
    dwarf_suppress_debuglink_crc, 213
Access to Section .debug_sup, 174
    dwarf_get_debug_sup, 175
Accessing a rnglists section, 275
Attaching a tied dbg, 240

Basic Library Datatypes Group, 35
    Dwarf_Addr, 36
    Dwarf_Bool, 36
    Dwarf_Half, 36
    Dwarf_Off, 35
    Dwarf_Ptr, 36
    Dwarf_Signed, 35
    Dwarf_Small, 36
    Dwarf_Unsigned, 35

Compilation Unit (CU) Access, 63
    dwarf_child, 68
    dwarf_cu_header_basics, 67
    dwarf_dealloc_die, 68
    dwarf_die_from_hash_signature, 69
    dwarf_find_die_given_sig8, 70
    dwarf_get_die_infotypes_flag, 70
    dwarf_next_cu_header_d, 65
    dwarf_next_cu_header_e, 64
    dwarf_offdie_b, 69
    dwarf_siblingof_b, 66
    dwarf_siblingof_c, 66

Debugging Information Entry (DIE) content, 71
    dwarf_addr_form_is_indexed, 74
    dwarf_arrayorder, 86
    dwarf_attr, 77
    dwarf_bitoffset, 85
    dwarf_bitsize, 84
    dwarf_bytesize, 84
    dwarf_CU_dieoffset_given_die, 75
    dwarf_debug_addr_index_to_addr, 74
    dwarf_die_abbrev_children_flag, 79
    dwarf_die_abbrev_code, 78
    dwarf_die_abbrev_global_offset, 72
    dwarf_die_CU_offset, 76
    dwarf_die_CU_offset_range, 76
    dwarf_die_offsets, 81
    dwarf_die_text, 77
    dwarf_diename, 78
    dwarf_dieoffset, 73
    dwarf_dietype_offset, 83
    dwarf_get_cu_die_offset_given_cu_header_offset_b, 75
    dwarf_get_die_address_size, 81
    dwarf_get_version_of_die, 82
    dwarf_hasattr, 80
    dwarf_highpc_b, 82
    dwarf_lowpc, 82
    dwarf_offset_list, 80
    dwarf_srclang, 85
    dwarf_tag, 73
    dwarf_validate_die_sibling, 79
Default stack frame #defines, 45
Defined and Opaque Structs, 38
    Dwarf_Abbrev, 43
    Dwarf_Arange, 44
    Dwarf_Attribute, 43
    Dwarf_Block, 39
    Dwarf_Cie, 43
    Dwarf_Cmdline_Options, 40
    Dwarf_Debug, 42
    Dwarf_Debug_Addr_Table, 42
    Dwarf_Die, 42
    Dwarf_Dnames_Head, 44
    Dwarf_Dsc_Head, 40
    Dwarf_Error, 42
    Dwarf_Fde, 43
    Dwarf_Form_Data16, 39
    Dwarf_Frame_Instr_Head, 40
    Dwarf_Gdbindex, 44
    Dwarf_Global, 43

Dwarf_Handler, 44
Dwarf_Line, 43
Dwarf_Line_Context, 44
Dwarf_Loc_Head_c, 40
Dwarf_Locdesc_c, 39
Dwarf_Macro_Context, 44
Dwarf_Obj_Access_Interface_a, 44
Dwarf_Obj_Access_Methods_a, 45
Dwarf_Obj_Access_Section_a, 45
dwarf_printf_callback_function_type, 40
Dwarf_Ranges, 40
Dwarf_Regtable3, 42
Dwarf_Regtable_Entry3, 41
Dwarf_Rnglists_Head, 45
Dwarf_Sig8, 39
Dwarf_Str_Offsets_Table, 40
Dwarf_Type, 43
Dwarf_Xu_Index_Header, 44
Demonstrating reading DWARF without a file., 276
Detaching a tied dbg, 240
Determine Object Type of a File, 237
DIE Attribute and Attribute-Form Details, 86
dwarf_attr_offset, 98
dwarf_attrlist, 88
dwarf_convert_to_global_offset, 99
dwarf_dealloc_attribute, 100
dwarf_dealloc_uncompressed_block, 99
dwarf_discr_entry_s, 101
dwarf_discr_entry_u, 101
dwarf_discr_list, 100
dwarf_formaddr, 93
dwarf_formblock, 96
dwarf_formdata16, 95
dwarf_formexprloc, 97
dwarf_formflag, 94
dwarf_formref, 90
dwarf_formsdata, 95
dwarf_formsig8, 92
dwarf_formsig8_const, 92
dwarf_formstring, 96
dwarf_formudata, 94
dwarf_get_debug_addr_index, 93
dwarf_get_debug_str_index, 97
dwarf_get_form_class, 98
dwarf_global_formref, 91
dwarf_global_formref_b, 91
dwarf_hasform, 88
dwarf_uncompress_integer_block_a, 98
dwarf_whatattr, 90
dwarf_whatform, 89
dwarf_whatform_direct, 89
Documenting Form_Block, 249
DW_DLA alloc/dealloc typename&number, 46
DW_DLE Dwarf_Error numbers, 47
DW_DLE_LAST, 56
DW_DLE_LAST
DW_DLE Dwarf_Error numbers, 56
Dwarf_Abbrev
Defined and Opaque Structs, 43
dwarf_add_debuglink_global_path
Access GNU .gnu_debuglink, build-id., 213
Dwarf_Addr
Basic Library Datatypes Group, 36
dwarf_addr_form_is_indexed
Debugging Information Entry (DIE) content, 74
Dwarf_Arange
Defined and Opaque Structs, 44
dwarf_arrayorder
Debugging Information Entry (DIE) content, 86
dwarf_attr
Debugging Information Entry (DIE) content, 77
dwarf_attr_offset
DIE Attribute and Attribute-Form Details, 98
Dwarf_Attribute
Defined and Opaque Structs, 43
dwarf_attrlist
DIE Attribute and Attribute-Form Details, 88
dwarf_basic_crc32
Access GNU .gnu_debuglink, build-id., 215
dwarf_bitoffset
Debugging Information Entry (DIE) content, 85
dwarf_bitsize
Debugging Information Entry (DIE) content, 84
Dwarf_Block
Defined and Opaque Structs, 39
Dwarf_Block_s, 285
Dwarf_Bool
Basic Library Datatypes Group, 36
dwarf_bytesize
Debugging Information Entry (DIE) content, 84
dwarf_check_lineheader_b
Line Table For a CU, 116
dwarf_child
Compilation Unit (CU) Access, 68
Dwarf_Cie
Defined and Opaque Structs, 43
dwarf_cie_section_offset
Stack Frame Access, 161
dwarf_close_str_offsets_table_access
Str_Offsets section details, 169
Dwarf_Cmdline_Options
Defined and Opaque Structs, 40
Dwarf_Cmdline_Options_s, 285
dwarf_convert_to_global_offset
DIE Attribute and Attribute-Form Details, 99
dwarf_crc32
Access GNU .gnu_debuglink, build-id., 214
dwarf_CU_dieoffset_given_die
Debugging Information Entry (DIE) content, 75
dwarf_cu_header_basics
Compilation Unit (CU) Access, 67
dwarf_dealloc
Generic dwarf_dealloc Function, 174
dwarf_dealloc_attribute
DIE Attribute and Attribute-Form Details, 100
dwarf_dealloc_debug_addr_table

.debug_addr access: DWARF5, 136
dwarf_dealloc_die
Compilation Unit (CU) Access, 68
dwarf_dealloc_dnames
Fast Access to .debug_names DWARF5, 177
dwarf_dealloc_error
Dwarf_Error Functions, 173
dwarf_dealloc_fde_cie_list
Stack Frame Access, 148
dwarf_dealloc_frame_instr_head
Stack Frame Access, 160
dwarf_dealloc_gdbindex
Fast Access to Gdb Index, 197
dwarf_dealloc_loc_head_c
Locations of data: DWARF2-DWARF5, 131
dwarf_dealloc_macro_context
Macro Access: DWARF5, 139
dwarf_dealloc_ranges
Ranges: code addresses in DWARF3-4, 119
dwarf_dealloc_rnglists_head
Rnglists: code addresses in DWARF5, 122
dwarf_dealloc_uncompressed_block
DIE Attribute and Attribute-Form Details, 99
dwarf_dealloc_xu_header
Fast Access to Split Dwarf (Debug Fission), 206
Dwarf_Debug
Defined and Opaque Structs, 42
dwarf_debug_addr_by_index
.debug_addr access: DWARF5, 135
dwarf_debug_addr_index_to_addr
Debugging Information Entry (DIE) content, 74
Dwarf_Debug_Addr_Table
Defined and Opaque Structs, 42
dwarf_debug_addr_table
.debug_addr access: DWARF5, 134
Dwarf_Debug_Fission_Per_CU_s, 285
Dwarf_Die
Defined and Opaque Structs, 42
dwarf_die_abbrev_children_flag
Debugging Information Entry (DIE) content, 79
dwarf_die_abbrev_code
Debugging Information Entry (DIE) content, 78
dwarf_die_abbrev_global_offset
Debugging Information Entry (DIE) content, 72
dwarf_die_CU_offset
Debugging Information Entry (DIE) content, 76
dwarf_die_CU_offset_range
Debugging Information Entry (DIE) content, 76
dwarf_die_from_hash_signature
Compilation Unit (CU) Access, 69
dwarf_die_offsets
Debugging Information Entry (DIE) content, 81
dwarf_die_text
Debugging Information Entry (DIE) content, 77
dwarf_diename
Debugging Information Entry (DIE) content, 78
dwarf_dieoffset
Debugging Information Entry (DIE) content, 73
dwarf_dietype_offset
Debugging Information Entry (DIE) content, 83
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, 100
dwarf_dnames_abbrevtable
Fast Access to .debug_names DWARF5, 177
dwarf_dnames_bucket
Fast Access to .debug_names DWARF5, 180
dwarf_dnames_cu_table
Fast Access to .debug_names DWARF5, 179
dwarf_dnames_entrypool
Fast Access to .debug_names DWARF5, 181
dwarf_dnames_entrypool_values
Fast Access to .debug_names DWARF5, 182
Dwarf_Dnames_Head
Defined and Opaque Structs, 44
dwarf_dnames_header
Fast Access to .debug_names DWARF5, 176
dwarf_dnames_name
Fast Access to .debug_names DWARF5, 180
dwarf_dnames_offsets
Fast Access to .debug_names DWARF5, 178
dwarf_dnames_sizes
Fast Access to .debug_names DWARF5, 178
Dwarf_Dsc_Head
Defined and Opaque Structs, 40
dwarf_errmsg
Dwarf_Error Functions, 172
dwarf_errmsg_by_number
Dwarf_Error Functions, 172
dwarf_errno
Dwarf_Error Functions, 171
Dwarf_Error
Defined and Opaque Structs, 42
Dwarf_Error Functions, 171
dwarf_dealloc_error, 173
dwarf_errmsg, 172
dwarf_errmsg_by_number, 172
dwarf_errno, 171
dwarf_error_creation, 172
dwarf_error_creation
Dwarf_Error Functions, 172
dwarf_expand_frame_instructions
Stack Frame Access, 157
Dwarf_Fde
Defined and Opaque Structs, 43
dwarf_fde_section_offset
Stack Frame Access, 160
dwarf_find_die_given_sig8
Compilation Unit (CU) Access, 70
dwarf_find_macro_value_start
Macro Access: DWARF2-4, 143
dwarf_finish
Libdwarf Initialization Functions, 61

Dwarf_Form_Class
Enumerators with various purposes, 37
Dwarf_Form_Data16
Defined and Opaque Structs, 39
Dwarf_Form_Data16_s, 286
dwarf_formaddr
DIE Attribute and Attribute-Form Details, 93
dwarf_formblock
DIE Attribute and Attribute-Form Details, 96
dwarf_formdata16
DIE Attribute and Attribute-Form Details, 95
dwarf_formexprloc
DIE Attribute and Attribute-Form Details, 97
dwarf_formflag
DIE Attribute and Attribute-Form Details, 94
dwarf_formref
DIE Attribute and Attribute-Form Details, 90
dwarf_formsdata
DIE Attribute and Attribute-Form Details, 95
dwarf_formsig8
DIE Attribute and Attribute-Form Details, 92
dwarf_formsig8_const
DIE Attribute and Attribute-Form Details, 92
dwarf_formstring
DIE Attribute and Attribute-Form Details, 96
dwarf_formudata
DIE Attribute and Attribute-Form Details, 94
Dwarf_Frame_Instr_Head
Defined and Opaque Structs, 40
Dwarf_Gdbindex
Defined and Opaque Structs, 44
dwarf_gdbindex_addressarea
Fast Access to Gdb Index, 199
dwarf_gdbindex_addressarea_entry
Fast Access to Gdb Index, 199
dwarf_gdbindex_culist_array
Fast Access to Gdb Index, 197
dwarf_gdbindex_culist_entry
Fast Access to Gdb Index, 197
dwarf_gdbindex_cuvector_inner_attributes
Fast Access to Gdb Index, 202
dwarf_gdbindex_cuvector_instance_expand_value
Fast Access to Gdb Index, 204
dwarf_gdbindex_cuvector_length
Fast Access to Gdb Index, 202
dwarf_gdbindex_header
Fast Access to Gdb Index, 196
dwarf_gdbindex_string_by_offset
Fast Access to Gdb Index, 204
dwarf_gdbindex_symboltable_array
Fast Access to Gdb Index, 201
dwarf_gdbindex_symboltable_entry
Fast Access to Gdb Index, 201
dwarf_gdbindex_types_culist_array
Fast Access to Gdb Index, 198
dwarf_gdbindex_types_culist_entry
Fast Access to Gdb Index, 198
dwarf_get_abbrev
Abbreviations Section Details, 164
dwarf_get_abbrev_children_flag
Abbreviations Section Details, 165
dwarf_get_abbrev_code
Abbreviations Section Details, 165
dwarf_get_abbrev_entry_b
Abbreviations Section Details, 166
dwarf_get_abbrev_tag
Abbreviations Section Details, 165
dwarf_get_address_size
Object Sections Data, 225
dwarf_get_arange
Fast Access to a CU given a code address, 184
dwarf_get_arange_cu_header_offset
Fast Access to a CU given a code address, 185
dwarf_get_arange_info_b
Fast Access to a CU given a code address, 185
dwarf_get_aranges
Fast Access to a CU given a code address, 183
dwarf_get_cie_augmentation_data
Stack Frame Access, 156
dwarf_get_cie_index
Stack Frame Access, 150
dwarf_get_cie_info_b
Stack Frame Access, 150
dwarf_get_cie_of_fde
Stack Frame Access, 149
dwarf_get_cu_die_offset
Fast Access to a CU given a code address, 185
dwarf_get_cu_die_offset_given_cu_header_offset_b
Debugging Information Entry (DIE) content, 75
dwarf_get_debug_addr_index
DIE Attribute and Attribute-Form Details, 93
dwarf_get_debug_str_index
DIE Attribute and Attribute-Form Details, 97
dwarf_get_debug_sup
Access to Section .debug_sup, 175
dwarf_get_debugfission_for_die
Fast Access to Split Dwarf (Debug Fission), 209
dwarf_get_debugfission_for_key
Fast Access to Split Dwarf (Debug Fission), 209
dwarf_get_die_address_size
Debugging Information Entry (DIE) content, 81
dwarf_get_die_infotypes_flag
Compilation Unit (CU) Access, 70
dwarf_get_die_section_name
Object Sections Data, 223
dwarf_get_die_section_name_b
Object Sections Data, 223
dwarf_get_EH_name
Names DW_TAG_member etc as strings, 219
dwarf_get_endian_copy_function
Miscellaneous Functions, 237
dwarf_get_fde_at_pc
Stack Frame Access, 155
dwarf_get_fde_augmentation_data
Stack Frame Access, 157
dwarf_get_fde_exception_info

Stack Frame Access, 149
dwarf_get_fde_for_die
Stack Frame Access, 155
dwarf_get_fde_info_for_all_regs3
Stack Frame Access, 152
dwarf_get_fde_info_for_all_regs3_b
Stack Frame Access, 151
dwarf_get_fde_info_for_cfa_reg3_b
Stack Frame Access, 154
dwarf_get_fde_info_for_cfa_reg3_c
Stack Frame Access, 154
dwarf_get_fde_info_for_reg3_b
Stack Frame Access, 153
dwarf_get_fde_info_for_reg3_c
Stack Frame Access, 152
dwarf_get_fde_instr_bytes
Stack Frame Access, 151
dwarf_get_fde_list
Stack Frame Access, 147
dwarf_get_fde_list_eh
Stack Frame Access, 147
dwarf_get_fde_n
Stack Frame Access, 155
dwarf_get_fde_range
Stack Frame Access, 148
dwarf_get_form_class
DIE Attribute and Attribute-Form Details, 98
dwarf_get_FORM_CLASS_name
Names DW_TAG_member etc as strings, 221
dwarf_get_frame_instruction
Stack Frame Access, 158
dwarf_get_frame_instruction_a
Stack Frame Access, 159
dwarf_get_FRAME_name
Names DW_TAG_member etc as strings, 220
dwarf_get_frame_section_name
Object Sections Data, 224
dwarf_get_frame_section_name_eh_gnu
Object Sections Data, 224
dwarf_get_globals
Fast Access to .debug_pubnames and more., 187
dwarf_get_globals_header
Fast Access to .debug_pubnames and more., 191
dwarf_get_gnu_index_block
Fast Access to GNU .debug_gnu_pubnames, 193
dwarf_get_gnu_index_block_entry
Fast Access to GNU .debug_gnu_pubnames, 194
dwarf_get_gnu_index_head
Fast Access to GNU .debug_gnu_pubnames, 192
dwarf_get_GNUIKIND_name
Names DW_TAG_member etc as strings, 219
dwarf_get_GNUIVIS_name
Names DW_TAG_member etc as strings, 220
dwarf_get_harmless_error_list
Harmless Error recording, 216
dwarf_get_line_section_name_from_die
Object Sections Data, 225
dwarf_get_LLEX_name
Names DW_TAG_member etc as strings, 220
dwarf_get_location_op_value_c
Locations of data: DWARF2-DWARF5, 130
dwarf_get_locdesc_entry_d
Locations of data: DWARF2-DWARF5, 129
dwarf_get_loclist_c
Locations of data: DWARF2-DWARF5, 128
dwarf_get_loclist_context_basics
Locations of data: DWARF2-DWARF5, 133
dwarf_get_loclist_head_basics
Locations of data: DWARF2-DWARF5, 132
dwarf_get_loclist_head_kind
Locations of data: DWARF2-DWARF5, 128
dwarf_get_loclist_lle
Locations of data: DWARF2-DWARF5, 133
dwarf_get_loclist_offset_index_value
Locations of data: DWARF2-DWARF5, 132
dwarf_get_MACINFO_name
Names DW_TAG_member etc as strings, 220
dwarf_get_macro_context
Macro Access: DWARF5, 137
dwarf_get_macro_context_by_offset
Macro Access: DWARF5, 138
dwarf_get_macro_defundef
Macro Access: DWARF5, 141
dwarf_get_macro_details
Macro Access: DWARF2-4, 144
dwarf_get_macro_import
Macro Access: DWARF5, 142
dwarf_get_MACRO_name
Names DW_TAG_member etc as strings, 220
dwarf_get_macro_op
Macro Access: DWARF5, 140
dwarf_get_macro_startend_file
Macro Access: DWARF5, 142
dwarf_get_offset_size
Object Sections Data, 225
dwarf_get_ranges_b
Ranges: code addresses in DWARF3-4, 119
dwarf_get_real_section_name
Object Sections Data, 223
dwarf_get_rnglist_context_basics
Rnglists: code addresses in DWARF5, 125
dwarf_get_rnglist_head_basics
Rnglists: code addresses in DWARF5, 125
dwarf_get_rnglist_offset_index_value
Rnglists: code addresses in DWARF5, 123
dwarf_get_rnglist_rle
Rnglists: code addresses in DWARF5, 126
dwarf_get_rnglists_entry_fields_a
Rnglists: code addresses in DWARF5, 121
dwarf_get_section_info_by_index
Object Sections Data, 227
dwarf_get_section_info_by_index_a
Object Sections Data, 227
dwarf_get_section_info_by_name
Object Sections Data, 226
dwarf_get_section_info_by_name_a

Object Sections Data, 226
dwarf_get_section_max_offsets_d
Object Sections Data, 229
dwarf_get_str
String Section .debug_str Details, 167
dwarf_get_tied_dbg
Libdwarf Initialization Functions, 63
dwarf_get_universalbinary_count
Miscellaneous Functions, 236
dwarf_get_version_of_die
Debugging Information Entry (DIE) content, 82
dwarf_get_xu_hash_entry
Fast Access to Split Dwarf (Debug Fission), 207
dwarf_get_xu_index_header
Fast Access to Split Dwarf (Debug Fission), 206
dwarf_get_xu_index_section_type
Fast Access to Split Dwarf (Debug Fission), 207
dwarf_get_xu_section_names
Fast Access to Split Dwarf (Debug Fission), 208
dwarf_get_xu_section_offset
Fast Access to Split Dwarf (Debug Fission), 208
Dwarf_Global
Defined and Opaque Structs, 43
dwarf_global_cu_offset
Fast Access to .debug_pubnames and more., 189
dwarf_global_die_offset
Fast Access to .debug_pubnames and more., 189
dwarf_global_formref
DIE Attribute and Attribute-Form Details, 91
dwarf_global_formref_b
DIE Attribute and Attribute-Form Details, 91
dwarf_global_name_offsets
Fast Access to .debug_pubnames and more., 190
dwarf_global_tag_number
Fast Access to .debug_pubnames and more., 190
dwarf_globals_by_type
Fast Access to .debug_pubnames and more., 188
dwarf_globals_dealloc
Fast Access to .debug_pubnames and more., 188
dwarf_globname
Fast Access to .debug_pubnames and more., 189
dwarf_gnu_debuglink
Access GNU .gnu_debuglink, build-id., 211
dwarf_gnu_index_dealloc
Fast Access to GNU .debug_gnu_pubnames, 193
Dwarf_Half
Basic Library Datatypes Group, 36
Dwarf_Handler
Defined and Opaque Structs, 44
dwarf_hasattr
Debugging Information Entry (DIE) content, 80
dwarf_hasform
DIE Attribute and Attribute-Form Details, 88
dwarf_highpc_b
Debugging Information Entry (DIE) content, 82
dwarf_init_b
Libdwarf Initialization Functions, 60
dwarf_init_path
Libdwarf Initialization Functions, 57
dwarf_init_path_a
Libdwarf Initialization Functions, 58
dwarf_init_path_dl
Libdwarf Initialization Functions, 58
dwarf_init_path_dl_a
Libdwarf Initialization Functions, 59
dwarf_insert_harmless_error
Harmless Error recording, 217
Dwarf_Line
Defined and Opaque Structs, 43
Dwarf_Line_Context
Defined and Opaque Structs, 44
dwarf_line_is_addr_set
Line Table For a CU, 113
dwarf_line_srcfileno
Line Table For a CU, 113
dwarf_lineaddr
Line Table For a CU, 114
dwarf_linebeginstatement
Line Table For a CU, 112
dwarf_lineblock
Line Table For a CU, 115
dwarf_lineendsequence
Line Table For a CU, 112
dwarf_lineno
Line Table For a CU, 113
dwarf_lineoff_b
Line Table For a CU, 114
dwarf_linesrc
Line Table For a CU, 115
dwarf_load_loclists
Locations of data: DWARF2-DWARF5, 131
dwarf_load_rnglists
Rnglists: code addresses in DWARF5, 123
Dwarf_Loc_Head_c
Defined and Opaque Structs, 40
Dwarf_Locdesc_c
Defined and Opaque Structs, 39
dwarf_loclist_from_expr_c
Locations of data: DWARF2-DWARF5, 130
dwarf_lowpc
Debugging Information Entry (DIE) content, 82
dwarf_machine_architecture
Object Sections Data, 228
Dwarf_Macro_Context
Defined and Opaque Structs, 44
dwarf_macro_context_head
Macro Access: DWARF5, 139
dwarf_macro_context_total_length
Macro Access: DWARF5, 138
Dwarf_Macro_Details_s, 286
dwarf_macro_operands_table
Macro Access: DWARF5, 140
dwarf_next_cu_header_d
Compilation Unit (CU) Access, 65
dwarf_next_cu_header_e
Compilation Unit (CU) Access, 64

dwarf_next_str_offsets_table
Str_Offsets section details, 169
Dwarf_Obj_Access_Interface_a
Defined and Opaque Structs, 44
Dwarf_Obj_Access_Interface_a_s, 286
Dwarf_Obj_Access_Methods_a
Defined and Opaque Structs, 45
Dwarf_Obj_Access_Methods_a_s, 287
Dwarf_Obj_Access_Section_a
Defined and Opaque Structs, 45
Dwarf_Obj_Access_Section_a_s, 287
dwarf_object_finish
Libdwarf Initialization Functions, 62
dwarf_object_init_b
Libdwarf Initialization Functions, 61
Dwarf_Off
Basic Library Datatypes Group, 35
dwarf_offdie_b
Compilation Unit (CU) Access, 69
dwarf_offset_list
Debugging Information Entry (DIE) content, 80
dwarf_open_str_offsets_table_access
Str_Offsets section details, 168
dwarf_package_version
Miscellaneous Functions, 234
dwarf_print_lines
Line Table For a CU, 117
dwarf_printf_callback_function_type
Defined and Opaque Structs, 40
Dwarf_Printf_Callback_Info_s, 288
dwarf_prologue_end_etc
Line Table For a CU, 116
Dwarf_Ptr
Basic Library Datatypes Group, 36
Dwarf_Ranges
Defined and Opaque Structs, 40
Dwarf_Ranges_Entry_Type
Enumerators with various purposes, 37
Dwarf_Ranges_s, 288
dwarf_record_cmdline_options
Miscellaneous Functions, 235
dwarf_register_printf_callback
Line Table For a CU, 118
Dwarf_Regtable3
Defined and Opaque Structs, 42
Dwarf_Regtable3_s, 288
Dwarf_Regtable_Entry3
Defined and Opaque Structs, 41
Dwarf_Regtable_Entry3_s, 289
dwarf_return_empty_pubnames
Fast Access to .debug_pubnames and more., 191
dwarf_rnglists_get_rle_head
Rnglists: code addresses in DWARF5, 121
Dwarf_Rnglists_Head
Defined and Opaque Structs, 45
dwarf_sec_group_map
Section Groups Objectfile Data, 232
dwarf_sec_group_sizes
Section Groups Objectfile Data, 231
dwarf_set_de_alloc_flag
Miscellaneous Functions, 235
dwarf_set_default_address_size
Miscellaneous Functions, 236
dwarf_set_frame_cfa_value
Stack Frame Access, 162
dwarf_set_frame_rule_initial_value
Stack Frame Access, 162
dwarf_set_frame_rule_table_size
Stack Frame Access, 161
dwarf_set_frame_same_value
Stack Frame Access, 162
dwarf_set_frame_undefined_value
Stack Frame Access, 163
dwarf_set_harmless_error_list_size
Harmless Error recording, 216
dwarf_set_reloc_application
Miscellaneous Functions, 234
dwarf_set_stringcheck
Miscellaneous Functions, 234
dwarf_set_tied_dbg
Libdwarf Initialization Functions, 62
dwarf_siblingof_b
Compilation Unit (CU) Access, 66
dwarf_siblingof_c
Compilation Unit (CU) Access, 66
Dwarf_Sig8
Defined and Opaque Structs, 39
Dwarf_Sig8_s, 289
Dwarf_Signed
Basic Library Datatypes Group, 35
Dwarf_Small
Basic Library Datatypes Group, 36
dwarf_srcfiles
Line Table For a CU, 103
dwarf_srclang
Debugging Information Entry (DIE) content, 85
dwarf_srclines_b
Line Table For a CU, 105
dwarf_srclines_comp_dir
Line Table For a CU, 107
dwarf_srclines_dealloc_b
Line Table For a CU, 106
dwarf_srclines_files_data_b
Line Table For a CU, 109
dwarf_srclines_files_indexes
Line Table For a CU, 109
dwarf_srclines_from_linecontext
Line Table For a CU, 105
dwarf_srclines_include_dir_count
Line Table For a CU, 110
dwarf_srclines_include_dir_data
Line Table For a CU, 110
dwarf_srclines_subprog_count
Line Table For a CU, 108
dwarf_srclines_subprog_data
Line Table For a CU, 108

dwarf_srclines_table_offset
Line Table For a CU, 107
dwarf_srclines_two_level_from_linecontext
Line Table For a CU, 106
dwarf_srclines_version
Line Table For a CU, 111
dwarf_str_offsets_statistics
Str_Offsets section details, 170
Dwarf_Str_Offsets_Table
Defined and Opaque Structs, 40
dwarf_str_offsets_value_by_index
Str_Offsets section details, 170
dwarf_suppress_debuglink_crc
Access GNU .gnu_debuglink, build-id., 213
dwarf_tag
Debugging Information Entry (DIE) content, 73
Dwarf_Type
Defined and Opaque Structs, 43
dwarf_uncompress_integer_block_a
DIE Attribute and Attribute-Form Details, 98
Dwarf_Unsigned
Basic Library Datatypes Group, 35
dwarf_validate_die_sibling
Debugging Information Entry (DIE) content, 79
dwarf_whatattr
DIE Attribute and Attribute-Form Details, 90
dwarf_whatform
DIE Attribute and Attribute-Form Details, 89
dwarf_whatform_direct
DIE Attribute and Attribute-Form Details, 89
Dwarf_Xu_Index_Header
Defined and Opaque Structs, 44

Enumerators with various purposes, 37
Dwarf_Form_Class, 37
Dwarf_Ranges_Entry_Type, 37
Examining Section Group data, 241
Example getting .debug_ranges data, 268
Example walking CUs(d), 246
Example walking CUs(e), 244
Extracting fde, cie lists., 264

Fast Access to .debug_names DWARF5, 175
dwarf_dealloc_dnames, 177
dwarf_dnames_abbrevtable, 177
dwarf_dnames_bucket, 180
dwarf_dnames_cu_table, 179
dwarf_dnames_entrypool, 181
dwarf_dnames_entrypool_values, 182
dwarf_dnames_header, 176
dwarf_dnames_name, 180
dwarf_dnames_offsets, 178
dwarf_dnames_sizes, 178
Fast Access to .debug_pubnames and more., 186
dwarf_get_globals, 187
dwarf_get_globals_header, 191
dwarf_global_cu_offset, 189
dwarf_global_die_offset, 189
dwarf_global_name_offsets, 190
dwarf_global_tag_number, 190
dwarf_globals_by_type, 188
dwarf_globals_dealloc, 188
dwarf_globname, 189
dwarf_return_empty_pubnames, 191
Fast Access to a CU given a code address, 183
dwarf_get_arange, 184
dwarf_get_arange_cu_header_offset, 185
dwarf_get_arange_info_b, 185
dwarf_get_aranges, 183
dwarf_get_cu_die_offset, 185
Fast Access to Gdb Index, 195
dwarf_dealloc_gdbindex, 197
dwarf_gdbindex_addressarea, 199
dwarf_gdbindex_addressarea_entry, 199
dwarf_gdbindex_culist_array, 197
dwarf_gdbindex_culist_entry, 197
dwarf_gdbindex_cuvector_inner_attributes, 202
dwarf_gdbindex_cuvector_instance_expand_value, 204
dwarf_gdbindex_cuvector_length, 202
dwarf_gdbindex_header, 196
dwarf_gdbindex_string_by_offset, 204
dwarf_gdbindex_symboltable_array, 201
dwarf_gdbindex_symboltable_entry, 201
dwarf_gdbindex_types_culist_array, 198
dwarf_gdbindex_types_culist_entry, 198
Fast Access to GNU .debug_gnu_pubnames, 192
dwarf_get_gnu_index_block, 193
dwarf_get_gnu_index_block_entry, 194
dwarf_get_gnu_index_head, 192
dwarf_gnu_index_dealloc, 193
Fast Access to Split Dwarf (Debug Fission), 205
dwarf_dealloc_xu_header, 206
dwarf_get_debugfission_for_die, 209
dwarf_get_debugfission_for_key, 209
dwarf_get_xu_hash_entry, 207
dwarf_get_xu_index_header, 206
dwarf_get_xu_index_section_type, 207
dwarf_get_xu_section_names, 208
dwarf_get_xu_section_offset, 208

Generic dwarf_dealloc Function, 173
dwarf_dealloc, 174

Harmless Error recording, 215
dwarf_get_harmless_error_list, 216
dwarf_insert_harmless_error, 217
dwarf_set_harmless_error_list_size, 216

LEB Encode and Decode, 233
Libdwarf Initialization Functions, 56
dwarf_finish, 61
dwarf_get_tied_dbg, 63
dwarf_init_b, 60
dwarf_init_path, 57
dwarf_init_path_a, 58
dwarf_init_path_dl, 58
dwarf_init_path_dl_a, 59

dwarf_object_finish, 62
dwarf_object_init_b, 61
dwarf_set_tied_dbg, 62
Line Table For a CU, 102
dwarf_check_lineheader_b, 116
dwarf_line_is_addr_set, 113
dwarf_line_srcfileno, 113
dwarf_lineaddr, 114
dwarf_linebeginstatement, 112
dwarf_lineblock, 115
dwarf_lineendsequence, 112
dwarf_lineno, 113
dwarf_lineoff_b, 114
dwarf_linesrc, 115
dwarf_print_lines, 117
dwarf_prologue_end_etc, 116
dwarf_register_printf_callback, 118
dwarf_srcfiles, 103
dwarf_srclines_b, 105
dwarf_srclines_comp_dir, 107
dwarf_srclines_dealloc_b, 106
dwarf_srclines_files_data_b, 109
dwarf_srclines_files_indexes, 109
dwarf_srclines_from_linecontext, 105
dwarf_srclines_include_dir_count, 110
dwarf_srclines_include_dir_data, 110
dwarf_srclines_subprog_count, 108
dwarf_srclines_subprog_data, 108
dwarf_srclines_table_offset, 107
dwarf_srclines_two_level_from_linecontext, 106
dwarf_srclines_version, 111
Location/expression access, 251
Locations of data: DWARF2-DWARF5, 126
dwarf_dealloc_loc_head_c, 131
dwarf_get_location_op_value_c, 130
dwarf_get_locdesc_entry_d, 129
dwarf_get_loclist_c, 128
dwarf_get_loclist_context_basics, 133
dwarf_get_loclist_head_basics, 132
dwarf_get_loclist_head_kind, 128
dwarf_get_loclist_lle, 133
dwarf_get_loclist_offset_index_value, 132
dwarf_load_loclists, 131
dwarf_loclist_from_expr_c, 130

Macro Access: DWARF2-4, 143
dwarf_find_macro_value_start, 143
dwarf_get_macro_details, 144
Macro Access: DWARF5, 136
dwarf_dealloc_macro_context, 139
dwarf_get_macro_context, 137
dwarf_get_macro_context_by_offset, 138
dwarf_get_macro_defundef, 141
dwarf_get_macro_import, 142
dwarf_get_macro_op, 140
dwarf_get_macro_startend_file, 142
dwarf_macro_context_head, 139
dwarf_macro_context_total_length, 138
dwarf_macro_operands_table, 140

Miscellaneous Functions, 233
dwarf_get_endian_copy_function, 237
dwarf_get_universalbinary_count, 236
dwarf_package_version, 234
dwarf_record_cmdline_options, 235
dwarf_set_de_alloc_flag, 235
dwarf_set_default_address_size, 236
dwarf_set_reloc_application, 234
dwarf_set_stringcheck, 234

Names DW_TAG_member etc as strings, 217
dwarf_get_EH_name, 219
dwarf_get_FORM_CLASS_name, 221
dwarf_get_FRAME_name, 220
dwarf_get_GNUIKIND_name, 219
dwarf_get_GNUIVIS_name, 220
dwarf_get_LLEX_name, 220
dwarf_get_MACINFO_name, 220
dwarf_get_MACRO_name, 220

Object Sections Data, 221
dwarf_get_address_size, 225
dwarf_get_die_section_name, 223
dwarf_get_die_section_name_b, 223
dwarf_get_frame_section_name, 224
dwarf_get_frame_section_name_eh_gnu, 224
dwarf_get_line_section_name_from_die, 225
dwarf_get_offset_size, 225
dwarf_get_real_section_name, 223
dwarf_get_section_info_by_index, 227
dwarf_get_section_info_by_index_a, 227
dwarf_get_section_info_by_name, 226
dwarf_get_section_info_by_name_a, 226
dwarf_get_section_max_offsets_d, 229
dwarf_machine_architecture, 228

Ranges: code addresses in DWARF3-4, 118
dwarf_dealloc_ranges, 119
dwarf_get_ranges_b, 119
Reading gdbindex addressarea, 269
Reading .debug_funcnames (nonstandard), 258
Reading .debug_macinfo (DWARF2-4), 263
Reading .debug_types (nonstandard), 259
Reading .debug_varnames data (nonstandard), 259
Reading .debug_weaknames (nonstandard), 258
Reading a location expression, 252
Reading an aranges section, 267
Reading cu and tu Debug Fission data, 271
Reading gdbindex data, 268
Reading high pc from a DIE., 272
Reading Split Dwarf (Debug Fission) data, 272
Reading Split Dwarf (Debug Fission) hash slots, 271
Reading string offsets section data, 266
Reading the .eh_frame section, 264
Reading the gdbindex symbol table, 270
Retrieving tag,attribute,etc names, 273
Rnglists: code addresses in DWARF5, 120
dwarf_dealloc_rnglists_head, 122
dwarf_get_rnglist_context_basics, 125

dwarf_get_rnglist_head_basics, 125
dwarf_get_rnglist_offset_index_value, 123
dwarf_get_rnglist_rle, 126
dwarf_get_rnglists_entry_fields_a, 121
dwarf_load_rnglists, 123
dwarf_rnglists_get_rle_head, 121

Section Groups Objectfile Data, 231
dwarf_sec_group_map, 232
dwarf_sec_group_sizes, 231
Stack Frame Access, 144
dwarf_cie_section_offset, 161
dwarf_dealloc_fde_cie_list, 148
dwarf_dealloc_frame_instr_head, 160
dwarf_expand_frame_instructions, 157
dwarf_fde_section_offset, 160
dwarf_get_cie_augmentation_data, 156
dwarf_get_cie_index, 150
dwarf_get_cie_info_b, 150
dwarf_get_cie_of_fde, 149
dwarf_get_fde_at_pc, 155
dwarf_get_fde_augmentation_data, 157
dwarf_get_fde_exception_info, 149
dwarf_get_fde_for_die, 155
dwarf_get_fde_info_for_all_regs3, 152
dwarf_get_fde_info_for_all_regs3_b, 151
dwarf_get_fde_info_for_cfa_reg3_b, 154
dwarf_get_fde_info_for_cfa_reg3_c, 154
dwarf_get_fde_info_for_reg3_b, 153
dwarf_get_fde_info_for_reg3_c, 152
dwarf_get_fde_instr_bytes, 151
dwarf_get_fde_list, 147
dwarf_get_fde_list_eh, 147
dwarf_get_fde_n, 155
dwarf_get_fde_range, 148
dwarf_get_frame_instruction, 158
dwarf_get_frame_instruction_a, 159
dwarf_set_frame_cfa_value, 162
dwarf_set_frame_rule_initial_value, 162
dwarf_set_frame_rule_table_size, 161
dwarf_set_frame_same_value, 162
dwarf_set_frame_undefined_value, 163
Str_Offsets section details, 168
dwarf_close_str_offsets_table_access, 169
dwarf_next_str_offsets_table, 169
dwarf_open_str_offsets_table_access, 168
dwarf_str_offsets_statistics, 170
dwarf_str_offsets_value_by_index, 170
String Section .debug_str Details, 167
dwarf_get_str, 167

Using dwarf_attrlist(), 248
Using dwarf_expand_frame_instructions, 265
Using dwarf_attrlist(), 239
Using dwarf_child(), 242
Using dwarf_discr_list(), 250
Using dwarf_get_globals(), 257
Using dwarf_globals_by_type(), 257
Using dwarf_init_path(), 238
Using dwarf_init_path_dl(), 238
Using dwarf_offdie_b(), 247
Using dwarf_offset_given_die(), 248
Using dwarf_offset_list(), 248
Using dwarf_siblingofb(), 242
Using dwarf_srcfiles(), 256
Using dwarf_srclines_b(), 253
Using dwarf_srclines_b() and linecontext, 256
using dwarf_validate_die_sibling, 243
Using GNU debuglink data, 273