z/OS



Common Debug Architecture Library Reference

SC09-7654-03

z/OS



Common Debug Architecture Library Reference

SC09-7654-03

Note
Before using this information and the product it supports, read the information in “Notices” on page 351.

Fourth Edition (September 2009)

This edition applies to Run-Time Library Extensions in IBM z/OS operating system Version 1 Release 11 (5694-A01), and to all subsequent releases until otherwise indicated in new editions.

This edition replaces SC09-7654-02. Make sure that you use the correct edition for the level of the program listed above. Also, ensure that you apply all necessary PTFs for the program.

IBM welcomes your comments. You can send your comments via e-mail to compinfo@ca.ibm.com. Be sure to include your e-mail address if you want a reply.

Include the title and order number of this document, and the page number or topic related to your comment. When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2004, 2009.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this document xiii
Who should use this document xiii
CDA and related publications xiii
Softcopy documents xv
Where to find more information xvi
Run-Time Library Extensions on the World Wide Web xvi
Information updates on the web xvi
Technical support xvi
How to send your comments xvi

Chapter 1. About Common Debug Architecture 1
CDA libraries and utilities 2
libelf 2
libdwarf 3
libddpi 3
isdcnvt 6
dwarfdump 7
Changes for CDA in z/OS V1R11 7
CDA requirements and recommendations 7
CDA limitations 8

Chapter 2. Ddpi_Init and Ddpi_Finish APIs 9
Ddpi_Info_Mode object 9
Ddpi_Info object 9
ddpi_init operation 10
ddpi_finish operation 11

Chapter 3. Ddpi_Error APIs 13
The libddpi error parameter 13
Ddpi_Error object 14
Ddpi_Handler object 14
Error-handling callback functions 14
ddpi_error_reset operation 15
ddpi_error_get_msg operation 15
ddpi_error_get_number operation 16
ddpi_error_get_errhandler operation 16
ddpi_error_set_errhandler operation 17
ddpi_error_get_errarg operation 18
ddpi_error_set_errarg operation 18
ddpi_error_show_error operation 19

Chapter 4. Processing storage deallocation APIs 21
Ddpi_StorageLocn object 21
Storage deallocation macros 21
ddpi_dealloc operation 22

Chapter 5. Ddpi_Addr APIs 23
Ddpi_Addr_Mode object 23
ddpi_addr_normalize operation 23
ddpi_addr_offset_normalize operation 24

Chapter 6. Ddpi_Elf loading API 25
ddpi_elf_load_cu operation 25

© Copyright IBM Corp. 2004, 2009

Chapter 7. Ddpi_Info APIs 27
ddpi_info_get_mode operation 27
ddpi_info_get_user_area operation 28
ddpi_info_list_space operation 28
ddpi_info_list_module operation 29
ddpi_info_set_dwarf_error_handler operation 30
ddpi_info_set_dbg_dirs operation 31

Chapter 8. Ddpi_Space APIs 33
Ddpi_Space object 33
Ddpi_ASID object 33
Ddpi_ALET object 33
Ddpi_Space_Type object 33
Ddpi_Xfer_Status object 34
Ddpi_GS_Handler callback function 34
Ddpi_SS_Handler object 35
ddpi_space_create operation 36
ddpi_space_term operation 37
ddpi_space_get_owner operation 38
ddpi_space_get_name operation 38
ddpi_space_set_name operation 39
ddpi_space_get_type operation 40
ddpi_space_get_asid operation 40
ddpi_space_set_asid operation 41
ddpi_space_get_alet operation 41
ddpi_space_set_alet operation 42
ddpi_space_get_limit operation 43
ddpi_space_set_limit operation 43
ddpi_space_get_user_area operation 44
ddpi_space_list_process operation 44
ddpi_space_list_hidden_module operation 45
ddpi_space_unhide_module operation 46
ddpi_space_is_hidden_module operation 47
ddpi_space_delete_module operation 48
ddpi_space_find_class operation 48

Chapter 9. Ddpi_Process APIs 51
Ddpi_PRID object 51
Ddpi_Process_Type object 51
Ddpi_Process object 51
ddpi_process_create operation 51
ddpi_process_term operation 52
ddpi_process_get_owner operation 53
ddpi_process_get_id operation 53
ddpi_process_set_id operation 54
ddpi_process_get_name operation 54
ddpi_process_set_name operation 55
ddpi_process_get_TCBaddr operation 56
ddpi_process_set_TCBaddr operation 56
ddpi_process_get_type operation 57
ddpi_process_get_user_area operation 58
ddpi_process_list_module operation 58
ddpi_process_hide_module operation 59
ddpi_process_list_class operation 60
ddpi_process_list_thread operation 60

Chapter 10. Ddpi_Thread APIs 63
Ddpi_THID object 63
Ddpi_Thread_Type object 63
Ddpi_Thread object 63
ddpi_thread_create operation 63
ddpi_thread_term operation 64
ddpi_thread_get_owner operation 65
ddpi_thread_get_name operation 66
ddpi_thread_set_name operation 66
ddpi_thread_get_id operation 67
ddpi_thread_set_id operation. 68
ddpi_thread_get_type operation. 68
ddpi_thread_get_TCBaddr operation 69
ddpi_thread_get_user_area operation 69
ddpi_thread_list_mutex operation 70
ddpi_thread_list_cond operation 71
ddpi_thread_list_lock operation 72
ddpi_thread_list_class operation 73

Chapter 11. Ddpi_Lock APIs 75
Ddpi_Lock object 75
Ddpi_LOID object 75
Ddpi_Lock_State object. 75
ddpi_lock_create operation 76
ddpi_lock_term operation 77
ddpi_lock_get_owner operation 77
ddpi_lock_get_name operation 78
ddpi_lock_set_name operation 78
ddpi_lock_get_id operation 79
ddpi_lock_set_id operation 80
ddpi_lock_get_state operation 80
ddpi_lock_set_state operation 81
ddpi_lock_get_user_area operation 81

Chapter 12. Ddpi_Mutex APIs 83
Ddpi_Mutex object 83
Ddpi_MUID object. 83
Ddpi_Mutex_State object 83
ddpi_mutex_create operation. 84
ddpi_mutex_term operation 84
ddpi_mutex_get_owner operation 85
ddpi_mutex_get_name operation 86
ddpi_mutex_set_name operation 86
ddpi_mutex_get_id operation. 87
ddpi_mutex_set_id operation. 88
ddpi_mutex_get_state operation 88
ddpi_mutex_set_state operation 89
ddpi_mutex_get_user_area operation 89

Chapter 13. Ddpi_Cond APIs 91
Ddpi_Cond object 91
Ddpi_CVID object 91
Ddpi_Cond object 91
ddpi_cond_create operation 91
ddpi_cond_term operation 92
ddpi_cond_get_owner operation 93

ddpi_cond_get_name operation . . . 93
ddpi_cond_set_name operation . . . 94
ddpi_cond_get_id operation . . . 95
ddpi_cond_set_id operation . . . 95
ddpi_cond_get_state operation . . . 96
ddpi_cond_set_state operation . . . 96
ddpi_cond_get_user_area operation . . . 97

Chapter 14. Ddpi_Module APIs . . . 99
Ddpi_Module object . . . 99
Ddpi_Module_Format object . . . 99
Ddpi_Module_Origin object . . . 100
Ddpi_Module_Owner_Type data type . . . 100
ddpi_module_create operation . . . 101
ddpi_module_term operation . . . 102
ddpi_module_get_access operation . . . 103
ddpi_module_get_owner operation . . . 103
ddpi_module_list_all_owners operation . . . 104
ddpi_module_get_major_name operation . . . 105
ddpi_module_get_minor_name operation . . . 106
ddpi_module_get_format operation . . . 107
ddpi_module_get_origin operation . . . 107
ddpi_module_get_usage operation . . . 108
ddpi_module_get_user_area operation . . . 108
ddpi_module_list_entrypt operation . . . 109
ddpi_module_list_class operation . . . 110
ddpi_module_find_space operation . . . 111
ddpi_module_extract_C_CPP_information operation . . . 112
ddpi_module_extract_debug_info operation . . . 113
ddpi_module_find_wsa operation . . . 114
ddpi_module_get_dwarf_error operation . . . 115
ddpi_module_list_function operation . . . 115
ddpi_module_list_variable operation . . . 117
ddpi_module_list_type operation . . . 118
ddpi_module_list_sourcefile operation . . . 119
ddpi_module_list_elf operation . . . 120
ddpi_module_find_elf_given_address operation . . . 120

Chapter 15. Ddpi_Access APIs . . . 123
Ddpi_Access object . . . 123
ddpi_access_create operation . . . 123
ddpi_access_term operation . . . 124
ddpi_access_get_owner operation . . . 125
ddpi_access_get_debug operation . . . 126
ddpi_access_set_debug operation . . . 126
ddpi_access_list_elf operation . . . 127
ddpi_access_get_user_area operation . . . 128
ddpi_access_get_dwarf_error operation . . . 129

Chapter 16. Ddpi_Elf APIs . . . 131
Ddpi_Elf object . . . 131
Ddpi_Elf_Source object . . . 131
Ddpi_Elf_Source_Type object . . . 132
ddpi_elf_create operation . . . 133
ddpi_elf_term operation . . . 134
ddpi_elf_get_owner operation . . . 135

ddpi_elf_get_source operation 135
ddpi_elf_set_source operation 136
ddpi_elf_get_elf operation 137
ddpi_elf_set_elf operation 137
ddpi_elf_get_elf_file_name operation 138
ddpi_elf_set_elf_file_name operation 138
ddpi_elf_get_ppa_addrs operation 139
ddpi_elf_set_ppa_addrs operation 140
ddpi_elf_get_md5_sig operation 141
ddpi_elf_set_md5_sig operation 142
ddpi_elf_get_csect_addrs operation 142
ddpi_elf_set_csect_addrs operation 143
ddpi_elf_get_user_area operation 144
ddpi_elf_list_function operation 144
ddpi_elf_list_variable operation 145
ddpi_elf_list_type operation 146
ddpi_elf_list_sourcefile operation 147
ddpi_elf_get_primary_sourcefile operation 148

Chapter 17. Ddpi_Class APIs 151
Ddpi_Class_Type object 151
ddpi_class_get_storage_attr operation 151
Ddpi_Class_Owner_Type object 152
Ddpi_Class_Owner object 153
Ddpi_Class object 153
ddpi_class_create operation 153
ddpi_class_term operation 154
ddpi_class_get_owner operation 155
ddpi_class_get_name operation 156
ddpi_class_set_name operation 156
ddpi_class_get_type operation. 157
ddpi_class_get_storage_attr operation 158
ddpi_class_get_addr_low operation 158
ddpi_class_get_addr_high operation 159
ddpi_class_get_user_area operation 159
ddpi_class_list_section operation 160

Chapter 18. Ddpi_Section APIs 163
Ddpi_Section opaque object 163
ddpi_section_create operation 163
ddpi_section_term operation 164
ddpi_section_get_owner operation 165
ddpi_section_get_name operation 165
ddpi_section_set_name operation 166
ddpi_section_get_addr_low operation 166
ddpi_section_get_addr_high operation 167
ddpi_section_set_addr operation 168
ddpi_section_get_user_area operation 168

Chapter 19. Ddpi_Function APIs 171
Ddpi_Function object 171
ddpi_function_get_full_name operation. 171
ddpi_function_get_short_name operation 172
ddpi_function_get_access operation. 172
ddpi_function_get_elf operation 173
ddpi_function_get_die_offset operation. 174

Chapter 20. Ddpi_Variable APIs 175
Ddpi_Variable object 175
ddpi_variable_get_full_name operation. 175
ddpi_variable_get_short_name operation 176
ddpi_variable_get_access operation. 176
ddpi_variable_get_die_offset operation. 177

Chapter 21. Ddpi_Type APIs 179
Ddpi_Type object 179
ddpi_type_get_access operation 179
ddpi_type_get_elf operation. 180
ddpi_type_get_die_offset operation 180

Chapter 22. Ddpi_Sourcefile APIs 183
Ddpi_Sourcefile object. 183
ddpi_sourcefile_get_full_name operation 183
ddpi_sourcefile_get_short_name operation 184
ddpi_sourcefile_get_access operation 184
ddpi_sourcefile_get_die_offset operation 185
ddpi_sourcefile_get_source_lines operation 186

Chapter 23. Ddpi_EntryPt APIs 187
Ddpi_EntryPt object. 187
Ddpi_EntryPt_Type object 187
ddpi_entrypt_create operation 187
ddpi_entrypt_term operation 189
ddpi_entrypt_get_owner operation 189
ddpi_entrypt_get_entry_name operation 190
ddpi_entrypt_set_entry_name operation 190
ddpi_entrypt_get_symbol_name operation 191
ddpi_entrypt_set_symbol_name operation 192
ddpi_entrypt_get_type operation 192
ddpi_entrypt_set_type operation 193
ddpi_entrypt_get_addr_mode operation 194
ddpi_entrypt_set_addr_mode operation 194
ddpi_entrypt_get_storage_extent operation 195
ddpi_entrypt_set_storage_extent operation 196
ddpi_entrypt_get_storage_offset operation 196
ddpi_entrypt_set_storage_offset operation 197
ddpi_entrypt_get_user_area operation 197

Chapter 24. Ddpi_Machinestate APIs 199
Ddpi_MachineState object 199
Ddpi_AR object 199
Ddpi_CR object 200
Ddpi_FPR object. 202
Ddpi_GPR object 203
Ddpi_PSW object 204
Ddpi_PSW_Type object 205
Ddpi_Context object 206
Ddpi_Context_Type object 206
ddpi_machinestate_create operation 206
ddpi_machinestate_term operation 207
ddpi_machinestate_init operation. 208
ddpi_machinestate_clone operation 209
ddpi_machinestate_copy operation 209

ddpi_machinestate_get_gpr operation 210
ddpi_machinestate_set_gpr operation 211
ddpi_machinestate_query_gpr_change operation 212
ddpi_machinestate_get_fpr operation 213
ddpi_machinestate_set_fpr operation 214
ddpi_machinestate_query_fpr_change operation 215
ddpi_machinestate_get_fpcr operation 215
ddpi_machinestate_set_fpcr operation 216
ddpi_machinestate_query_fpcr_change operation 217
ddpi_machinestate_get_ar operation 218
ddpi_machinestate_set_ar operation 219
ddpi_machinestate_query_ar_change operation 220
ddpi_machinestate_get_cr operation 220
ddpi_machinestate_set_cr operation 221
ddpi_machinestate_query_cr_change operation 222
ddpi_machinestate_get_psw operation 223
ddpi_machinestate_set_psw operation 224
ddpi_machinestate_query_psw_change operation 225
ddpi_machinestate_get_ip operation 226
ddpi_machinestate_set_ip operation 227
ddpi_machinestate_get_amode operation 227
ddpi_machinestate_set_amode operation 228
ddpi_machinestate_get_space operation 229
ddpi_machinestate_set_space operation 230
ddpi_machinestate_get_storagelocn operation 230
ddpi_machinestate_get_context operation 231
ddpi_machinestate_set_context operation 232
ddpi_machinestate_reset_change operation 232
ddpi_machinestate_any_change operation 233

Chapter 25. Ddpi_PPA_Extract APIs 235
Ddpi_PPA1_data_struct_s object 235
Ddpi_PPA2_data_struct_s object 236
ddpi_ppa_extract_ppa1 operation 236
ddpi_ppa_extract_ppa2 operation 237
ddpi_ppa_extract_ppa1_entrypoint_name operation 238
ddpi_ppa_extract_CU_primary_source operation 239
ddpi_ppa2_md5_sig operation 240

Chapter 26. Ddpi_StackState APIs 243
Ddpi_StackState object 243
Ddpi_Stack_Format object 244
Ddpi_Stack_Type object 244
Ddpi_Stack_Linkage object 245
Ddpi_StackState_Identify_Handler object 246
Ddpi_StackState_Parent_Handler object 246
ddpi_stackstate_create operation 247
ddpi_stackstate_term operation 247
ddpi_stackstate_init operation 248
ddpi_stackstate_identify operation 249
ddpi_stackstate_parent operation 250
ddpi_stackstate_get_stack_format operation 251
ddpi_stackstate_set_stack_format operation 252
ddpi_stackstate_get_frame_type operation 252
ddpi_stackstate_set_frame_type operation 253
ddpi_stackstate_get_linkage operation 253

ddpi_stackstate_set_linkage operation 254
ddpi_stackstate_get_dsa_locn operation 255
ddpi_stackstate_set_dsa_locn operation 255
ddpi_stackstate_get_dsa_len operation 256
ddpi_stackstate_set_dsa_len operation 257
ddpi_stackstate_get_laa_locn operation 257
ddpi_stackstate_set_laa_locn operation 258
ddpi_stackstate_get_parent_locn operation 258
ddpi_stackstate_set_parent_locn operation 259
ddpi_stackstate_get_alloca_base operation 260
ddpi_stackstate_set_alloca_base operation 260
ddpi_stackstate_get_ep_locn operation 261
ddpi_stackstate_set_ep_locn operation 261
ddpi_stackstate_get_ppa1_locn operation 262
ddpi_stackstate_set_ppa1_locn operation 263
ddpi_stackstate_get_ppa2_locn operation 263
ddpi_stackstate_set_ppa2_locn operation 264

Chapter 27. Ddpi_StackState_Fn APIs 265
Ddpi_StackState_Fn object 265
ddpi_stackstate_fn_create operation 265
ddpi_stackstate_fn_term operation 266
ddpi_stackstate_fn_add operation 266
ddpi_stackstate_fn_get_count operation 267
ddpi_stackstate_fn_get_identify operation 268
ddpi_stackstate_fn_set_identify operation. 268
ddpi_stackstate_fn_get_parent operation 269
ddpi_stackstate_fn_set_parent operation 270

Chapter 28. Operations for Language Environment linkages 271
ddpi_stackstate_identify_le operation 271
ddpi_stackstate_parent_le operation 271

Chapter 29. Ddpi_Storage APIs. 273
Transparent access. 274
Opaque access 274
Ddpi_SL_Policy_Trans object 274
Ddpi_SStor_Token object 275
Ddpi_StorageLocn object. 275
Ddpi_SavedStorage object 275
ddpi_storagelocn_create operation 276
ddpi_storagelocn_term operation 277
ddpi_storagelocn_get_space operation. 277
ddpi_storagelocn_set_space operation. 278
ddpi_storagelocn_get_addr operation 278
ddpi_storagelocn_set_addr operation 279
ddpi_storagelocn_get_storage operation 280
ddpi_storagelocn_set_storage operation 281
ddpi_storagelocn_get_policy operation. 282
ddpi_storagelocn_set_policy operation. 282
ddpi_storagelocn_get_user_area operation 283
ddpi_savedstorage_create operation 284
ddpi_savedstorage_term operation 285
ddpi_savedstorage_term_all operation 285
ddpi_savedstorage_list_all_tokens operation 286
ddpi_savedstorage_list_modified operation 287

ddpi_savedstorage_get operation 288
ddpi_savedstorage_list_all operation 289
ddpi_savedstorage_find operation 289
ddpi_savedstorage_next operation 290
ddpi_savedstorage_get_modified operation 291
ddpi_savedstorage_set_modified operation 292
ddpi_savedstorage_reset_change operation 293
ddpi_savedstorage_dump operation 293

Chapter 30. Ddpi_Format APIs 295
Ddpi_Format specifiers 295
ddpi_formatter operation 296
ddpi_format_address operation 297
ddpi_format_bitfield_address operation 298
ddpi_format_set_input_charset operation 299
ddpi_format_set_type_format operation 300
ddpi_format_get_type_format operation 300
ddpi_format_set_composite_format operation 301
ddpi_format_get_composite_format operation 302
ddpi_format_expand_type_format operation 303
ddpi_format_i_to_hex operation 303
ddpi_format_hexdump operation 304
ddpi_format_chardump operation 305
ddpi_format_dbx_hexdump operation 306
ddpi_format_get_DIE_xeval_token operation 307
ddpi_format_get_DIE_member operation 308
ddpi_format_get_array_DIE_xeval_token operation 309
ddpi_format_get_ptr_to_mem_xeval_token operation 311
ddpi_format_set_showbases operation 312
ddpi_format_set_maxstring operation 312
ddpi_format_set_expand_classes operation 312
ddpi_format_set_expand_structs operation 313
ddpi_format_set_expand_unions operation 313
ddpi_format_set_expand_enums operation 313
ddpi_format_set_expand_funcpar operation 314
ddpi_format_set_expand_memfunc operation 314
ddpi_format_set_expand_memdata operation 314
ddpi_format_clear_user_format operation 315

Chapter 31. Ddpi_Xeval APIs 317
Ddpi_Xeval_Xtended_Op object 317
Ddpi_Xeval_Token_Kind object 318
Ddpi_Xeval_Token object 318
Ddpi_Xeval_Context object 318
Ddpi_Xeval_Unary_Func object 319
Ddpi_Xeval_Binary_Func object 320
ddpi_xeval_eval_unary_op operation 320
ddpi_xeval_eval_binary_op operation 322
ddpi_xeval_override_conv_func operation 323
ddpi_xeval_override_unary_func operation 324
ddpi_xeval_override_binary_func operation 325
ddpi_xeval_provide_unary_func_user_type_support operation 325
ddpi_xeval_provide_binary_func_user_type_support operation 326
ddpi_xeval_engine operation 327

Chapter 32. Code set specification APIs 329

ddpi_info_set_codeset operation 329
ddpi_format_set_codeset operation 330

Chapter 33. Character translation APIs 333
ddpi_translate_ibm1047_to_iso8859_1 operation 333
ddpi_translate_iso8859_1_to_ibm1047 operation 334

Chapter 34. Code set conversion APIs 335
ddpi_convert_c_cpp_isdobj operation 335
ddpi_fp_convert_c_cpp_isdobj operation 335

Chapter 35. Build information APIs 337
ddpi_build_version operation 337
ddpi_dll_version operation 337

Appendix A. libddpi error macros and messages 339
DDPI_DLE_LAST error macro 339
Error messages 339

Appendix B. Accessibility 349
Using assistive technologies 349
Keyboard navigation of the user interface. 349
z/OS information 349

Notices 351
Programming interface information 352
Trademarks. 352
Standards 353

Index 355

About this document

This information is the reference for the Common Debug Architecture (CDA) library `libddpi`. It provides a brief overview of CDA. The majority of the document is a detailed description of every CDA API. CDA is written in the C programming language and is based on ELF ABIs and the DWARF format.

This document uses the following terminology:

API *Application programming interface*. An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program. An extension to a standard DWARF API can include:

- Extensions to standard DWARF files, objects, or operations
- Additional objects or operations

object In object-oriented design or programming, a concrete realization (instance) of a class that consists of data and the operations associated with that data. An object contains the instance data that is defined by the class, but the class owns the operations that are associated with the data. Objects described in this document are generally a type definition or data structure, a container for a callback function prototype, or items that have been added to a DWARF file.

operation
In object-oriented design or programming, a service that can be requested *at the boundary of an object*. Operations can modify an object or disclose information about an object.

Who should use this document

This document is intended for programmers who will be developing program analysis applications and debugging applications for the IBM® XL C/C++ compiler on the IBM z/OS® operating system. The libraries provided by CDA allow applications to create or look for DWARF debugging information from ELF object files on the z/OS V1R10 operating system.

This document is a reference rather than a tutorial. It assumes that you have a working knowledge of the following items:

- The z/OS operating system
- The `libdwarf` APIs
- The `libelf` APIs
- The ELF ABI
- Writing debugging programs in C on z/OS
- Writing debugging programs in XL C/C++ on z/OS
- POSIX on z/OS
- The IBM Language Environment® on z/OS
- UNIX® System Services shell on z/OS

CDA and related publications

This section summarizes the content of the CDA publications and shows where to find related information in other publications.

© Copyright IBM Corp. 2004, 2009

Table 1. CDA, DWARF, and ELF publications

Document title and number	Key sections/chapters in the document
z/OS Common Debug Architecture User's Guide, SC09-7653	The user's guide for the `libddpi` library. It includes: • Overview of the `libddpi` architecture. • Information on the order and purpose of calls to `libddpi` operations used to access DWARF information on behalf of model user applications. • Hints for using CDA with C/C++ source. See http://www.ibm.com/software/awdtools/libraryext/library/.
z/OS DWARF/ELF Extensions Library Reference, SC09-7655	The reference for IBM extensions to the `libdwarf` and `libelf` libraries. It includes: • Extensions to `libdwarf` consumer APIs (Chapters 2 through 8) • Extensions to `libdwarf` producer APIs (Chapters 9 through 19) • Extensions to `libelf` APIs and utilities (Chapter 20) This document discusses only these extensions, and does not provide a detailed explanation of DWARF and ELF. See http://www.ibm.com/software/awdtools/libraryext/library/.
System V Application Binary Interface Standard	The Draft April 24, 2001 version of the ELF standard. For more information, go to: http://www.ibm.com/software/awdtools/libraryext/library/.
ELF Application Binary Interface Supplement	The Draft April 24, 2001 version of the ELF standard supplement. For more information, go to: http://www.ibm.com/software/awdtools/libraryext/library/.
DWARF Debugging Information Format, Version 3	The Draft 8 (November 19, 2001) version of the DWARF standard. This document is available on the web.
Consumer Library Interface to DWARF	The revision 1.48, March 31, 2002, version of the `libdwarf` consumer library. See http://www.ibm.com/software/awdtools/libraryext/library/.
Producer Library Interface to DWARF	The revision 1.18, January 10, 2002, version of the `libdwarf` producer library. See http://www.ibm.com/software/awdtools/libraryext/library/.
MIPS Extensions to DWARF Version 2.0	The revision 1.17, August 29, 2001, version of the MIPS extension to DWARF. See http://www.ibm.com/software/awdtools/libraryext/library/.
z/OS XL C/C++ User's Guide, SC09-4767	Guidance information for: • z/OS C/C++ examples • Compiler options • Binder options and control statements • Specifying z/OS Language Environment run-time options • Compiling, IPA linking, binding, and running z/OS C/C++ programs • Utilities (Object Library, CXXFILT, DSECT Conversion, Code Set and Locale, ar and make, BPXBATCH, c89, xlc, as, CDAHLASM) • Diagnosing problems • Cataloged procedures and REXX™ EXECs supplied by IBM See http://www.ibm.com/software/awdtools/czos/library.

Table 1. CDA, DWARF, and ELF publications (continued)

Document title and number	Key sections/chapters in the document
z/OS XL C/C++ Programming Guide, SC09-4767	Guidance information for: • Implementing programs that are written in C and C++ • Developing C and C++ programs to run under z/OS • Using XPLINK assembler in C and C++ applications • Debugging I/O processes • Using advanced coding techniques, such as threads and exception handlers • Optimizing code • Internationalizing applications

The following table lists the related publications for CDA, ELF, and DWARF. The table groups the publications according to the tasks they describe.

Table 2. Publications by task

Tasks	Documents
Coding programs	• *DWARF/ELF Extensions Library Reference, SC09-7655* • *z/OS Common Debug Architecture Library Reference, SC09-7654* • *z/OS Common Debug Architecture User's Guide, SC09-7653* • *DWARF Debugging Information Format* • *Consumer Library Interface to DWARF* • *Producer Library Interface to DWARF* • *MIPS Extensions to DWARF Version 2.0*
Compiling, binding, and running programs	• *z/OS XL C/C++ User's Guide, SC09-4767* • *z/OS XL C/C++ Programming Guide, SC09-4765*
General discussion of CDA	• *z/OS Common Debug Architecture User's Guide, SC09-7653* • *z/OS Common Debug Architecture Library Reference, SC09-7654*
Environment and application APIs (objects and operations)	• *z/OS Common Debug Architecture Library Reference, SC09-7654*
A guide to using the libraries	• *z/OS Common Debug Architecture Library Reference, SC09-7654*
Examples of producer and consumer programs	• *z/OS Common Debug Architecture User's Guide, SC09-7653*

Softcopy documents

The following information describes where you can find softcopy documents.

The IBM z/OS Common Debug Architecture publications are supplied in PDF formats and IBM BookMaster® formats on the following CD: *z/OS Collection, SK3T-4269*. They are also available at the following Web site: http://www.ibm.com/software/awdtools/libraryext/library

To read a PDF file, use the Adobe® Reader. If you do not have the Adobe Reader, you can download it (subject to Adobe license terms) from the Adobe web site at www.adobe.com.

You can also browse the documents on the World Wide Web by visiting the z/OS library at http:// www.ibm.com/servers/eserver/zseries/zos/bkserv/.

Note: For further information on viewing and printing softcopy documents and using IBM BookManager®, see *z/OS Information Roadmap*.

Where to find more information

Please see *z/OS Information Roadmap* for an overview of the documentation associated with IBM z/OS.

Run-Time Library Extensions on the World Wide Web

Additional information on Common Debug Architecture is available on the World Wide Web on the Run-Time Library Extensions home page at: http://www.ibm.com/software/awdtools/libraryext/

This page contains links to other useful information, including the Run-Time Library Extensions information library, which includes the Common Debug Architecture documents.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and Documentation APARs for IBM z/OS, refer to the online list of APARs and PTFs. This document is updated weekly and lists documentation changes before they are incorporated into z/OS publications.

The online list of APARs and PTFs is found at:http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/ZIDOCMST/CCONTENTS

Technical support

Additional technical support is available from the IBM z/OS XL C/C++ Support page. This page provides a portal with search capabilities to a large selection of technical support FAQs and other support documents.

You can find the z/OS XL C/C++ Support page on the Web at: http://www.ibm.com/software/awdtools/czos/support.

If you cannot find what you need, you can e-mail:

compinfo@ca.ibm.com

For the latest information about z/OS XL C/C++, visit the product information site at: http://www.ibm.com/software/awdtools/czos/

For information about boosting performance, productivity and portability, visit the C/C++ Cafe at: http://www-949.ibm.com/software/rational/cafe/community/ccpp.

How to send your comments

Your feedback is important in helping to provide accurate and high-quality information. If you have any comments about this document or the IBM z/OS XL C/C++ documentation, send your comments by e-mail to: compinfo@ca.ibm.com

Be sure to include the name of the document, the part number of the document, the version of, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Chapter 1. About Common Debug Architecture

Common Debug Architecture (CDA) was introduced in z/OS V1R5 to provide a consistent format for debug information on z/OS. As such, it provides an opportunity to work towards a common debug information format across the various languages and operating systems that are supported on the IBM zSeries® eServer™ platform. The product is implemented in the z/OS CDA libraries component of the z/OS Run-Time Library Extensions element of z/OS (V1R5 and higher).

CDA components are based on:

- "The DWARF industry-standard debugging information format"
- "Executable and Linking Format (ELF) application binary interfaces (ABIs)"

CDA-compliant applications store DWARF debugging information in separate ELF object files, rather than a typical object file. When the executable module is loaded into memory.

- The size of the executable module is reduced.
- Memory usage is minimized.

The DWARF industry-standard debugging information format

The DWARF 3 debugging format is an industry-standard format developed by the UNIX International Programming Languages Special Interest Group (SIG). It is designed to meet the symbolic, source-level debugging needs of different languages in a unified fashion by supplying language-independent debugging information. The debugging information format is open-ended, allowing for the addition of debugging information that accommodates new languages or debugger capabilities.

DWARF was developed by the UNIX International Programming Languages Special Interest Group (SIG).

The use of DWARF has two distinct advantages:

- It provides a stable and maintainable debug information format for all languages.
- It facilitates porting program analysis and debug applications to z/OS from other DWARF-compliant platforms.

Executable and Linking Format (ELF) application binary interfaces (ABIs)

Using a separate ELF object file to store debugging information enables the program analysis application to load specific information only as it is needed. You use the DEBUG option of the z/OS XL C/C++ compiler to create the separate ELF object file, which has a `*.dbg` extension.

Note: In this information, those ELF object files may be referred to as an ELF object file, an ELF object, or an ELF file. Such a file stores only DWARF debugging information.

© Copyright IBM Corp. 2004, 2009

CDA libraries and utilities

CDA comprises three libraries and two utilities.

The libraries are:

- `libelf`
- `libdwarf`
- `libddpi`

The utilities are:

- `isdcnvt`
- `dwarfdump`

To ensure compatibility, the `libdwarf` and `libelf` libraries are packaged together in a single DLL.

The `libddpi` library is available as a dynamic linking library, as well as an static library in XPLINK object form. It is highly recommended that you use the DLL version of `libddpi` when creating your application.

Regardless of whether a 64-bit or 31-bit version of a library is used, the created information is binary-equivalent. For example, the z/OS XL C/C++ compilers could use a 31-bit version of `libdwarf` and `libelf` to create the debug information, although the z/OS dbx debugger always uses a 64-bit version of `libdwarf`, `libelf` and `libddpi` when reading the debug information.

The `libelf` and `libdwarf` libraries each consist of two sets of APIs:

- A set of producer APIs that help create ELF/DWARF
- A set of consumer APIs that help users of ELF/DWARF

The `libdwarf/libelf` DLL contains all producer and consumer APIs.

libelf

The `libelf` APIs are used to create the ELF descriptor. The descriptor is then used by other APIs to read from, and write to, the ELF object file.

`libelf` is packaged as part of a dynamic link library (DLL). Both the 31-bit version and the 64-bit version are packaged as part of `CEE.SCEERUN2`.

- For 64–bit applications, `libelf` is shipped in the `CDAEQED DLL` as part of `CEE.SCEERUN2`.
- For 31–bit applications, `libelf` is shipped in the `CDAEED DLL` as part of `CEE.SCEERUN2`.

When compiling an application that uses the `libelf` library, you must:

- Include `libelf.h`
- Bind the module with an appropriate side deck:
  - For 64–bit applications:
    - Bind with CEE.SCEELIB(CDAEQED) if you are using an IBM MVS™ file system
    - Bind with `/usr/lpp/cbclib/lib/libelfdwarf64.x` if you are using a hierarchical file system
  - For 31–bit applications, use either of the following:

- Bind with CEE.SCEELIB(CDAEED) if you are using an MVS file system
- Bind with /usr/lpp/cbclib/lib/libelfdwarf32.x if you are using a hierarchical file system

Note: IBM has extended the libelf library to support C/C++ on the z/OS operating system. These extensions enable the libelf library to be used in various environments without additional extensions. The generic interfaces provided by libelf are defined as part of the UNIX System V Release 4 ABI. For descriptions of the interfaces supported by libelf, refer to the following documents:

- *System V Application Binary Interface Standard*
- *DWARF/ELF Extensions Library Reference*

libdwarf

The libdwarf APIs:

- Create or read DWARF objects that include DWARF debugging information
- Use ELF descriptors to read from, and write to, the ELF object file

libdwarf is packaged as a dynamic link library (DLL). Both the 31-bit version and the 64-bit version are packaged as part of CEE.SCEERUN2:

- For 64–bit applications, libdwarf is shipped in the CDAEQED DLL
- For 31–bit applications, libdwarf is shipped in the CDAEED DLL

When compiling an application that uses the libdwarf library, you must:

- Include both libdwarf.h and dwarf.h (which are located in the /usr/lpp/cbclib/include/libdwarf directory)
- Bind the module with an appropriate side deck:
  - For 64–bit applications:
    - Bind with CEE.SCEELIB(CDAEQED) if you are using an MVS file system
    - Bind with /usr/lpp/cbclib/lib/libelfdwarf64.x if you are using a hierarchical file system
  - For 31–bit applications:
    - Bind with CEE.SCEELIB(CDAEED) if you are using an MVS file system
    - Bind with /usr/lpp/cbclib/lib/libelfdwarf32.x if you are using a hierarchical file system

Note: IBM has extended the libdwarf library to support C/C++ on the z/OS operating system. The IBM extensions to libdwarf provide:

- Improved speed and memory utilization
- Support for the IBM zSeries eServer C/C++ languages

For information that is specific to these extensions, see *DWARF/ELF Extensions Library Reference*.

libddpi

The Debug Data Program Information library (libddpi) provides a repository for gathering information about a program module. A debugger or other program analysis application can use the repository to collect and query information from the program module.

libddpi:

- Supports conversion of non-DWARF C/C++ debugging information to the DWARF format. For example, the `libddpi` library is used to convert In Store Debug (ISD) information.
- Puts an environmental context around the DWARF information for both the producer APIs and the consumer APIs. For information on how to use `libddpi`, see *Common Debug Architecture User's Guide*. This document provides the library reference information for `libddpi`.

The `libddpi` library is packaged as the static library `libddpi.a` in the `/usr/lpp/cbclib/lib` directory. This directory contains both the 31-bit and 64-bit versions of the library.

The `libddpi` library is also packaged as a dynamic link library (DLL). Both the 31-bit version and the 64-bit version are packaged as part of CEE.SCEERUN2:

- For 64-bit applications, `libddpi` is shipped in the CDAEQDPI DLL.
- For 31-bit applications, `libddpi` is shipped in the CDAEDPI DLL.

When creating or compiling an application that uses `libddpi`, you must:

- Include `libddpi.h` in your source code

  The `libddpi.h` file is located in the `/usr/lpp/cbclib/include/libddpi/` directory.
- Bind the module with an appropriate side deck:
  - For 64-bit applications:
    - Bind with CEE.SCEELIB(CDAEQDPI) if you are using an MVS file system
    - Bind with `/usr/lpp/cbclib/lib/libddpi64.x` if you are using a hierarchical file system.
  - For 31-bit applications:
    - Bind with CEE.SCEELIB(CDAEDPI) if you are using an MVS file system.
    - Bind with `/usr/lpp/cbclib/lib/libddpi32.x` if you are using a hierarchical file system.

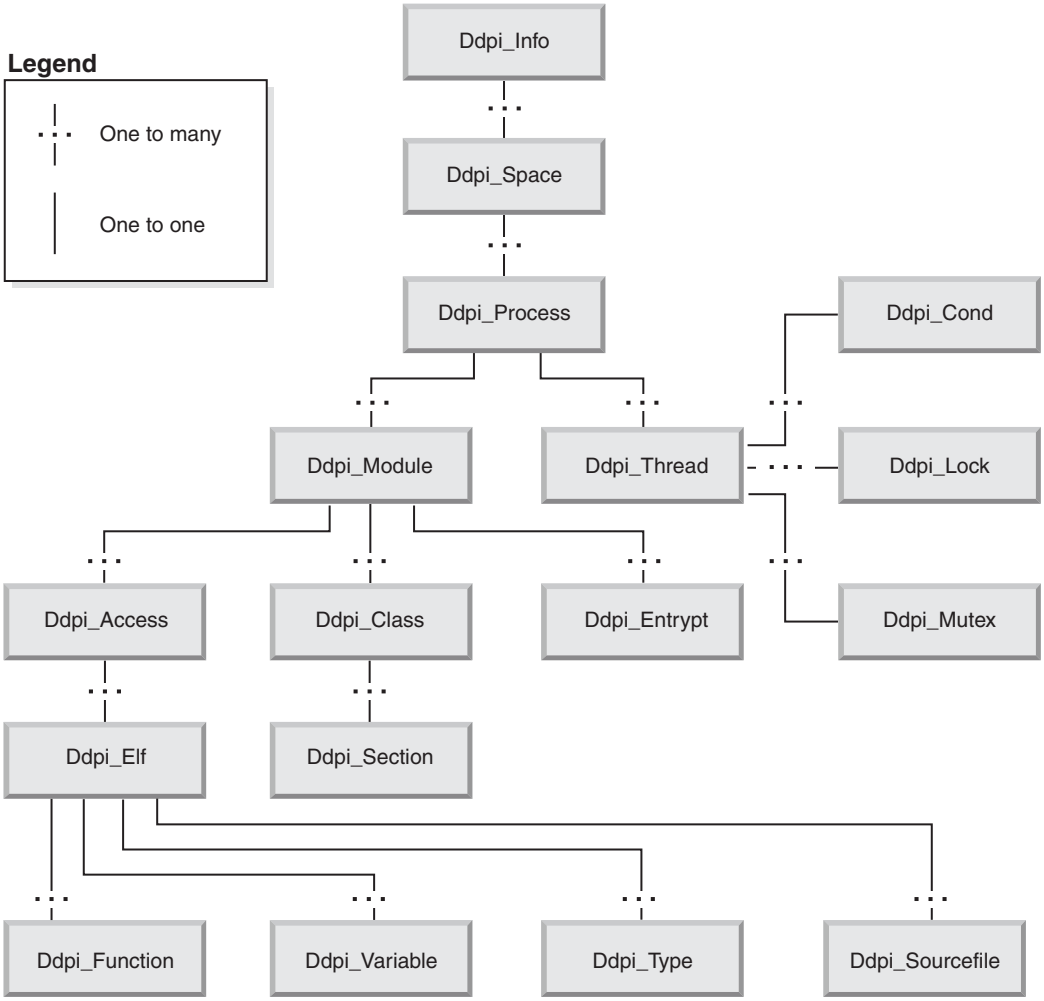
The main groups of APIs in `libddpi` are described in the following table:

API groups	Description
CDA application model APIs: • Ddpi_Init and Ddpi_Finish APIs • Ddpi_Error APIs • Processing storage deallocation APIs • Ddpi_Addr APIs • Ddpi_Elf loading API • Ddpi_Info APIs • Ddpi_Space APIs • Ddpi_Process APIs • Ddpi_Thread APIs • Ddpi_Lock APIs • Ddpi_Mutex APIs • Ddpi_Cond APIs • Ddpi_Module APIs • Ddpi_Access APIs • Ddpi_Elf APIs • Ddpi_Class APIs • Ddpi_Section APIs • Ddpi_EntryPt APIs	This group of consumer and producer APIs allows developers to model applications they are analyzing and to use those models to keep track of debugging information.
CDA APIs that support use of the module map: • Ddpi_Function APIs • Ddpi_Variable APIs • Ddpi_Type APIs • Ddpi_Source APIs	The operations in this group: • Find and extract information about a specific function, including static functions. Each `Ddpi_Function` object is owned by a `Ddpi_Elf object`. A `ddpi_function` operation queries one or more `Ddpi_Function` objects and extracts information about the specific function. • Provide information about global variables. Each `Ddpi_Variable` object is owned by a `Ddpi_Elf object`. • Provide information about external types. Each `Ddpi_Type` object is owned by a `Ddpi_Elf object`. • Provide information about source files. Each `Ddpi_Source` object is owned by a `Ddpi_Elf object`.
System-dependent APIs	This group provides system-specific helper APIs.
System-independent APIs	This group provides generic common helper APIs.
DWARF-expression APIs	This group provides a DWARF expression evaluator which assists with the evaluation of some of the DWARF opcodes.
Utilities	This group: • Helps convert ISD debugging information into DWARF debugging information. • supports the integrity of the program analysis application build.

CDA application model APIs

CDA application model APIs allow the CDA user to model the program that is being analyzed and track the debug information through the use of that model.

The following high-level diagram shows how some of the APIs in the CDA application mode relate to one another:



Note: This diagram does not show all the possible relationships within the hierarchy.

isdcnvt

Note: `isdcnvt` cannot be used to convert 64-bit objects. Debug information for 64–bit XL C/C++ applications is available only in DWARF format.

`isdcnvt` is a stand-alone utility that converts objects with In Store Debug (ISD) information into an ELF object file. In other words, `isdcnvt` accepts objects with ISD C/C++ debugging information and generates an ELF object file containing debugging information in the DWARF format. It is shipped in the `/usr/lpp/cbclib/bin/isdcnvt` directory.

This converter supports debugging information generated by the TEST option for XL C/C++ compilers. For more information, see “CDA limitations” on page 8.

The following restrictions apply to the `isdcnvt` utility:

- Debugging information cannot be converted if the compilation unit (CU) has only line number information. This occurs if the GONUMBER and NOTEST compiler options are used.
- CUs cannot be converted if they have data only and do not contain any functions.

The required ISD information is generated by the IBM XL C/C++ compiler TEST option.

For more information on `isdcnvt`, see *Common Debug Architecture User's Guide*. For more information on the code set conversion APIs, see Chapter 34, “Code set conversion APIs,” on page 335.

dwarfdump

The `dwarfdump` utility displays the debugging information of an ELF object file in user-readable form. It is shipped in the `/usr/lpp/cbclib/bin` directory.

`dwarfdump` works on DWARF objects nested within an ELF container. It can be used to validate the work of a developer who is accessing and manipulating DWARF debugging information.

To display help information, run `dwarfdump` without arguments.

Changes for CDA in z/OS V1R11

CDA libraries shipped with IBM z/OS V1R11 include an additional API which can return the contents of the source file at specified line numbers. For detailed information about this API, see “ddpi_sourcefile_get_source_lines operation” on page 186.

CDA requirements and recommendations

The CDA libraries are compiled with the z/OS XL C/C++ compiler.

To provide flexibility for developers who want to use the CDA application model, many `libddpi` objects have a variable-length user area. This allows the developers to store their own extra information in the `libddpi` model.

When you use CDA libraries, be aware of the following requirements and recommendations:

- If your application uses CDA libraries or utilities, you must compile it with the XPLINK compiler option.
- To ensure the best possible application performance, run applications with the HEAPPOOLS(on) run-time option.
  - For 31-bit applications, you must specify the HEAPPOOLS(on) option in a pragma or CEEUOPT.
  - For 64-bit applications, the HEAPPOOLS(on) option is the default.
- Notice the code set in which strings are accepted and returned. By default, most character strings accepted and returned by the CDA libraries are encoded in the

ISO8859-1 code set. You can use code set conversion operations to change the code set. For more information about the z/OS XL C/C++ compiler options, see *z/OS XL C/C++ User's Guide, SC09-4767*.

CDA limitations

When you use CDA libraries, be aware of the following limitations:

- Conversion support for ISD debugging information is available only for 31-bit object files, modules or program objects built with:
  - IBM C/C++ for MVS/ESA™ V3R2
  - Any release of z/OS XL C/C++

  This support is not intended to work with debugging information generated by the IBM C/370™ or IBM AD/Cycle® C/370 compilers.

  The CDA converter will be updated to match the TEST option support for the version of z/OS with which it is shipping. However, a lower-level CDA converter might not be able to properly convert the debugging data generated by the TEST option on a newer level of the z/OS C/C++ compiler.
- You must gather information and call the appropriate `libddpi` interface to generate objects (such as `Ddpi_Space` and `Ddpi_Process`) that can be used to model the behavior of an application under analysis. Although the `libddpi` library contains these objects, they are not created automatically when the application triggers an event.

  Note: These `libddpi` objects were created to:
  - Provide a structured information repository in a common format
  - Allow CDA to use expanded queries across a whole application, whether or not the application information is in an ELF object file, or has been modelled using `libddpi` elements such as `Ddpi_Section`

Chapter 2. Ddpi_Init and Ddpi_Finish APIs

The initialization and terminations APIs provide the data objects and functional operations required to initialize and terminate `libddpi` instances.

The `Ddpi_Info_Mode` object is a transparent data type that contains situation information (for example, the scope of the information and whether the data is acceptable to the `ddpi_init`) operation.

The `Ddpi_Info` object is:

- An opaque data type that stores the information that a program generates about itself as it runs (that is, internal session information for the current`libddpi` instance).
- The global structure (that is, the base object) for the application model.

Most `ddpi` operations take `Ddpi_Info` as a parameter either directly or indirectly.

You can use `ddpi_init` to initialize multiple `libddpi` instances. Each `libddpi` instance creates its own `Ddpi_Info` object.

Ddpi_Info_Mode object

`Ddpi_Info_Mode` determines the type of environment being analyzed by `ddpi` initialization and termination functions.

Ddpi_Info_Mode type definition

```
typedef enum Ddpi_Info_Mode_s {
  Ddpi_IM_Unknown   = 0,
  Ddpi_IM_Machine   = 1,
  Ddpi_IM_Mod_File  = 2,
  Ddpi_IM_Cu_File   = 3
} Ddpi_Info_Mode;
```

Ddpi_Info_Mode members

Ddpi_IM_Unknown
If the value is 0, it will not be accepted by `ddpi_init`.

Ddpi_IM_Machine
If the value is 1, this information processing session is on an active machine or in a post-mortem situation.

Ddpi_IM_Mod_File
If the value is 2, this information-processing session is on a processing utility for one or more module files (for example, on a module-level debugger).

Ddpi_IM_Cu_File
If the value is 3, this information is being processed at the compile unit (CU) or object file level. For example, the standalone conversion utility (`isdcnvt`) is a sample CU processor. The utility converts debugging information from the in-store debugging (ISD) format to the DWARF format.

Ddpi_Info object

The `Ddpi_Info` object is an opaque data type that stores the information that a program generates about itself as it runs.

© Copyright IBM Corp. 2004, 2009

A program analysis application can gather this information and store it in a Ddpi_Info object. The program analysis application can then use the Ddpi_Info contents in its analysis of the program.

Examples of a program analysis application are:

- Debuggers and debug binders
- Profilers
- CU conversions

Type definition

```
typedef struct Ddpi_Info_s*
```

ddpi_init operation

The ddpi_init operation creates a Ddpi_Info object, which manages an libddpi processing session.

The Ddpi_Info object must be created before any other libddpi API is called.

The ddpi_init operation has two error-handling parameters:

- errhand defines the callback function that is called when an error occurs.
- errarg is a pointer that is passed to errhand when it is called.

.

The ddpi_init operation initializes any existing Ddpi_Error objects.

Note: If you do not terminate any previously used Ddpi_Error objects with ddpi_finish, the memory it uses cannot be reallocated.

Prototype

```
int ddpi_init(
  Ddpi_info_Mode        mode,
  Ddpi_Handler          errhand,
  Dwarf_Ptr             errarg,
  int                   user_area_len,
  Ddpi_info*            ret_info,
  Ddpi_Error*           error);
```

Parameters

mode
This input parameter accepts the Ddpi_Info processing mode.

errhand
Input. This parameter accepts either the error-handling callback function or NULL.

errarg
Input. This parameter accepts either a pointer to additional information or NULL. It can be used to pass extra information to errhand. The pointer address, and not its target value, is copied into libddpi storage. The pointer can be specified when either the ddpi_init or ddpi_error_set_errarg operation is called. errarg can be queried and changed using ddpi_error operations.

user_area_len
This input parameter accepts the user-area length.

ret_info
This output parameter returns the Ddpi_Info object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful completion of the initialization.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if any of the following are true:

- The mode parameter is invalid. For more information, see “Parameters” on page 10.
- ret_info is full.
- user_area_len is less than zero.
- An error occurs during allocation.

ddpi_finish operation

This operation terminates the current libddpi processing session and releases all resources associated with the Ddpi_Info descriptor.

Both the Ddpi_Info descriptor and the given Ddpi_Error object are freed and invalidated. The ddpi_finish operation then triggers the deallocation of any remaining entities that were children of the given Ddpi_Info descriptor.

Prototype

```
int ddpi_finish(
    Ddpi_Info            info,
    Ddpi_Error*          error);
```

Parameters

info
Input. This input parameter accepts the Ddpi_Info processing object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful release of resources associated with the Ddpi_Info descriptor.

DW_DLV_ERROR
This value is returned if:

- info is NULL
- An unexpected error occurs while freeing storage

Note: The ddpi_finish operation never returns DW_DLV_NO_ENTRY.

Chapter 3. Ddpi_Error APIs

Ddpi_Error APIs provide a consistent method of detecting and setting error conditions.

A Ddpi_Error object is created in response to an error condition generated by a producer or consumer application. Most ddpi operations take a Ddpi_Error object as a parameter. The exceptions are the utility operations and some flag-setting ddpi format operations.

When a call to a ddpi operation returns a value of DW_DLV_ERROR, an error has occurred. Error codes are set within the Ddpi_Error object to notify the program analysis application of the error condition. To extract error information from the error object, the program analysis application needs to contain error-handling callback functions.

If the Ddpi_Error object indicates that the error condition was triggered by a call to a libdwarf operation, the program analysis application can extract further information by querying the Dwarf_Error object that is stored in the Ddpi_Access object. For more information, see Chapter 15, “Ddpi_Access APIs,” on page 123.

If invalid arguments are specified when a ddpi operation is called, the return values are undefined. Because the return values specify the error-handling callback function to be called, the ddpi operation might terminate abnormally.

Examples of invalid arguments are:

- A NULL pointer to a ddpi operation (except where explicitly permitted).
- A pointer to an invalid address.
- A pointer to an uninitialized value.

The libddpi error parameter

The error parameter is a required parameter for all ddpi operations. It accepts and returns either the Ddpi_Error object or NULL.

Before a Ddpi_Error object can be used to capture error situations, it must be passed as an error parameter when the ddpi_init operation is called. In other words:

- For every libddpi instance, a unique Ddpi_Error object is initialized when the error parameter is passed to the ddpi_init operation.
- When an libddpi instance is terminated with theddpi_finish operation, the program analysis application must pass the corresponding Ddpi_error object as a parameter. The ddpi_finish operation terminates both the Ddpi_Info object and its corresponding Ddpi-Error object.

When an error occurs, the error parameter value determines how the error condition is handled:

- If the error parameter is not NULL, error information is stored in the Ddpi_Error object.
- If the error parameter is NULL:
  - The libddpi error process looks for the error-handling callback function specified by the ddpi_init operation.
  - If no callback function was specified, the operation terminates abnormally.

© Copyright IBM Corp. 2004, 2009

Ddpi_Error object

The Ddpi_Error object contains the most recent libddpi error code.

There are no creation or termination operations for Ddpi_Error objects. The space for the object is allocated when ddpi_init is called. When a Ddpi_Error object is created, it is initialized to NULL. The space for the object is deallocated when ddpi_finish is called.

Note: If any Ddpi_Error object is generated, the Ddpi_Error type definition applies to the entire instance.

Type definition

```
typedef struct Ddpi_Error_s*         Ddpi_Error;
```

Ddpi_Handler object

You must create your own error-handling procedure. The Ddpi_Handler object gives the type and interface information for this procedure. During initialization of the Ddpi_Info object, the callback function is given by the errhand parameter. After initialization, a Ddpi_Handler object is created whenever an error occurs, unless a Ddpi_Error object was passed to the function when it was called.

Type definition

```
typedef void  (*Ddpi_Handler)(
  Ddpi_Error            error,
  Dwarf_Ptr             errarg);
```

Members

error
Input/Output. This accepts and returns the error parameter, which is a required parameter that handles error information generated by the ddpi producer or consumer operation. If error is not NULL, error information is stored in the given object. If error is NULL, the libddpi error process looks for an error-handling callback function specified by the ddpi_init operation. If no callback function was specified, the error process aborts.

errarg
Input. This accepts a pointer to additional information, as specified by the user for the program analysis application. The pointer can be specified when either the ddpi_init or the ddpi_error_set_errarg operation is called.

Error-handling callback functions

An error-handling callback function and a pointer to an error argument (errarg) can be specified by the program analysis application during initialization of libddpi. During initialization of the Ddpi_Info object, the callback function is given by the errhand parameter. If libddpi detects an error and no Ddpi_Error object is specified in the application, the error-handling callback function is called.

Note: You must create your own error-handling callback function. For more information, see ″Writing DWARF data to the ELF object file″ in *z/OS Common Debug Architecture User's Guide*.

After initialization, the program analysis application can change the either the callback function or the errarg pointer received from the callback function. The

callback function is changed by calling the ddpi_error_set_errhandler operation. The pointer is changed by calling the ddpi_error_set_errarg operation.

If an error-handling callback function has been specified in the application and if the value of the error parameter is NULL, the error parameter is passed to the specified callback function, and the operation that encountered the error condition returns a value of DW_DLV_ERROR.

Callback function arguments

The callback function passes two arguments:

- The first argument contains a pointer to a temporary Ddpi_Error object that documents the error.
- The second argument is the errarg pointer, which is provided as a convenience for the program analysis application. The pointer can be used to pass extra information to the callback function.

If invalid arguments are specified when an libddpi operation is called, the return values are undefined. The error-handling callback function might not be called, and the libddpi operation might abort execution. Examples of invalid arguments are:

- A NULL pointer to a libddpi operation (except where explicitly permitted)
- A pointer to an invalid address
- A pointer to an uninitialized value

ddpi_error_reset operation

The ddpi_error_reset operation resets the Ddpi_Error object value to the DDPI_DLE_NO_ERROR macro.

Unallocated Ddpi_Error objects are not used.

Prototype

```
int  ddpi_error_reset(        Ddpi_Error*         error);
```

Parameters

error
See “The libddpi error parameter” on page 13.

Return values

The ddpi_error_reset operation always returns DW_DLV_OK.

ddpi_error_get_msg operation

The ddpi_error_get_msg operation retrieves the text message that describes the current error code.

These error messages are available only in English, and are provided only for your convenience when you are debugging an application. For a list of possible error messages, see Appendix A, “libddpi error macros and messages,” on page 339.

Prototype

```
char*  ddpi_error_get_msg(
         Ddpi_Error          error);
```

Parameters

error
See “The libddpi error parameter” on page 13.

Return values

The `ddpi_error_get_msg` operation always returns an error message string. If the given `error` is NULL, this operation returns a message stating that the `error` is NULL. If the given `error` contains an unrecognized error code, the operation will return a message stating that the `Ddpi_Error` value is out of range.

ddpi_error_get_number operation

This operation retrieves the current error number as a `Dwarf_Unsigned` object.

The returned error number corresponds to one of the given DDPI_DLE macros given in libddpi.h. For more information about DDPI_DLE macros, see Appendix A, “libddpi error macros and messages,” on page 339.

Note: Validating the number is the responsibility of the caller.

Prototype

```
Dwarf_Unsigned  ddpi_error_get_number(
                Ddpi_Error     error);
```

Parameters

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful retrieval of the error number.

DW_DLV_ERROR
If error is not NULL, the operation returns the error value without verifying that it is within the valid range. Validating the number is the responsibility of the caller.

DW_DLV_NO_ENTRY
This value is returned if the value of the `error` parameter is NULL.

ddpi_error_get_errhandler operation

This operation designates the given error-handling callback function as the `libddpi` error handler.

Prototype

```
int  ddpi_error_get_errhandler(
     Ddpi_Info           info,
     Ddpi_Handler*       ret_errhand,
     Ddpi_Error *        error);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

ret_errhand
This input parameter accepts either the error-handling callback function or NULL. `ret_errhand` can be queried and changed using Ddpi_Error operations.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful retrieval of the error handling callback function.

DW_DLV_NO_ENTRY
This value is returned if the `ret_errhand` is NULL.

DW_DLV_ERROR
This value is returned if:

- `info` is NULL
- `ret_errhand` is NULL
- An error occurs while allocating memory

ddpi_error_set_errhandler operation

This operation designates the given error-handling callback function as the `libddpi` error handler.

Prototype

```
int Ddpi_Handler   ddpi_error_set_errhandler(
    Ddpi_Info      info,
    Ddpi_Handler   errhand,
    Ddpi_Error *   error);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

errhand
Input. This parameter accepts either the error-handling callback function or NULL.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful release of resources associated with the `Ddpi_Info` descriptor.

DW_DLV_NO_ENTRY
This value is returned if the value of the `error` parameter is NULL.

DW_DLV_ERROR
If the value of the `error` parameter is not NULL, the operation returns the error value without verifying that it is within the valid range. This value is returned if:

- `info` is NULL
- An unexpected error occurs while freeing storage

ddpi_error_get_errarg operation

This operation assigns the given `error` argument to the `libddpi error` argument.

Prototype

```
Ddpi_Handler  ddpi_error_get_errarg(
     Ddpi_Info           info,
     Dwarf_Ptr*          ret_errarg,
     Ddpi_Error *        error);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

ret_errarg
Output. This parameter returns the current `error` argument.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
This value is returned upon successful assignment of the given error argument to the `libddpi` error argument.

DW_DLV_ERROR
This value is returned if info is NULL.

Note: The `ddpi_error_get_errarg` operation never returns DW_DLV_NO_ENTRY.

ddpi_error_set_errarg operation

This operation assigns the given `error` argument to the `libddpi error` argument.

Prototype

```
int  ddpi_error_set_errarg(
     Ddpi_Info           info,
     Dwarf_Ptr          errarg);
     Ddpi_Error *        error);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

errarg
Input. This parameter accepts either a pointer to additional information or NULL. It can be used to pass extra information to `errhand`. The pointer address, and not its target value, is copied into `libddpi` storage. The pointer can be specified when either the `ddpi_init` or `ddpi_error_set_errarg` operation is called. `errarg` can be queried and changed using `ddpi_error` operations.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
This value is returned upon successful assignment of the given error argument to the `libddpi` error argument.

DW_DLV_NO_ENTRY.
Never returned.

DW_DLV_ERROR
This value is returned if info is NULL.

ddpi_error_show_error operation

This operation enables or disables the verbose error display.

Error messages are controlled by the error handler provided by the user. By default, the verbose error display is disabled (false) and messages are not sent to STDERR. If the verbose error display is enabled (true), error messages are sent to STDERR whenever an error is detected. The message includes the message number, text, and traceback (if available).

Prototype

```
int ddpi_error_show_error(
    Ddpi_Info             info,
    Dwarf_Bool            new_show,
    Dwarf_Bool*           ret_prev_show,
    Ddpi_Error*           error);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

new_show
Enables (0) or disables (1) the verbose error display.

ret_prev_show
This output parameter returns the previous setting for the error display.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful enablement or disablement of the verbose error display.

DW_DLV_ERROR
This value is returned if:

- `info` is NULL
- `ret_prev_show` is NULL

Chapter 4. Processing storage deallocation APIs

The libddpi uses its own memory management processes when creating and terminating libddpi objects. In addition, some operations allocate memory on behalf of the program analysis (user) application. The ddpi_dealloc operation frees memory that was allocated on behalf of an application.

Examples of how to use ddpi_dealloc are given for each operation that requires it. For example, the ddpi_access_list_elf operation allocates memory in order to list the Ddpi_Elf objects associated with the given Ddpi_Access object. The code that deletes this list is:

```
ddpi_dealloc (info, *ret_elfs, DDPI_DLA_LIST)
```

where ret_elfs is the returned list.

Use the ddpi_dealloc operation only when it is recommended in the header file. The effect of using this operation on unknown pointers is undefined.

Ddpi_StorageLocn object

Ddpi_storagelocn is a pointer to data in memory, and not the actual data itself. It is an opaque data type that represents the address of a given location within an address space that is targeted by the user.

The Ddpi_storagelocn object contains:

- The address space, in the form of a “Ddpi_Space object” on page 33.
- The location address within the address space.
- The access policy, either opaque or transparent, to be used when accessing the storage.
- The user area, which is an optional object-extension area, and may be used by the Ddpi_StorageLocn object caller.

The creation, initialization, and destruction of a Ddpi_storagelocn object are handled as follows:

- The object is created by a successful call to the ddpi_storagelocn_create operation.
- The object is initialized by a successful call to the ddpi_storagelocn_init operation.
- Memory is deallocated by a successful call to the ddpi_storagelocn_term operation.

Type definition

```
typedef struct Ddpi_StorageLocn_s * Ddpi_StorageLocn;
```

Storage deallocation macros

Storage deallocation macros define the storage-deallocation types for use by program analysis applications. These values are specific to the type field of the ddpi_dealloc operation.

DDPI_DLA_LIST
This is used for an array of Dwarf_Ptr.

© Copyright IBM Corp. 2004, 2009

DDPI_DLA_ADDR
This is used for an array of Dwarf_Addr.

DDPI_DLA_ERROR
This is used for an array of Ddpi_Error objects.

DDPI_DLA_CHUNK
This is used for a chunk of bytes.

DDPI_DLA_STRING
This is used for an array of characters.

DDPI_DLA_SSTOR_TOKEN
This is used for an array of Ddpi_SSTor_Token objects.

ddpi_dealloc operation

This operation frees memory that has been allocated in support of the user. This operation is used to deallocate lists, entry points, DIEs and tokens.

Always deallocate `libddpi` entities by using their corresponding termination functions.

Notes:

1. Do not use the `free` function to deallocate the memory.
2. Do not use the `ddpi_dealloc` operation to deallocate `libddpi` entities such as the `Ddpi_Info` object.

Prototype

```
int ddpi_dealloc(
    Ddpi_Info            info,
    void*                space,
    int                  type);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

space
Input. This accepts the “Ddpi_StorageLocn object” on page 21.

type
Input. This accepts the storage allocation type. See “Storage deallocation macros” on page 21

Return values

DW_DLV_OK
This value is returned upon successful deallocation of memory allocated to the user area.

DW_DLV_ERROR
This value is returned if `info` is NULL.

DW_DLV_NO_ENTRY
This value is returned if there is no storage to free, or if the given type is not known.

Chapter 5. Ddpi_Addr APIs

The Ddpi_Addr APIs are common routines that help you work with zSeries address types.

Ddpi_Addr_Mode object

The Ddpi_Addr_Mode object accepts valid constants that specify the environment's addressing mode.

Type definition

```
typedef enum Ddpi_Addr_Mode_s {
  Ddpi_AM_Unknown = 0,
  Ddpi_AM_24      = 1,
  Ddpi_AM_31      = 2,
  Ddpi_AM_64      = 3
} Ddpi_Addr_Mode;
```

Ddpi_Addr_Mode members

Ddpi_AM_Unknown
If the accepted value is 0, the addressing mode is not known and the Ddpi_Addr APIs will return DW_DLV_ERROR.

Ddpi_AM_24
If the accepted value is 1, the ddpi_addr operations will use a 24-bit addressing mode.

Ddpi_AM_31
If the accepted value is 2, the ddpi_addr operations will use a 31-bit addressing mode.

Ddpi_AM_64
If the accepted value is 3, the ddpi_addr operations will use a 64-bit addressing mode.

ddpi_addr_normalize operation

This operation analyzes a Dwarf address to ensure that it is in the correct range for the given addressing mode. It also removes extraneous data.

Prototype

```
int ddpi_addr_normalize(
  Ddpi_Info             info,
  Dwarf_Addr            in_addr,
  Ddpi_Addr_Mode        amode,
  Dwarf_Addr*           ret_addr,
  Ddpi_Error*           error);
```

Parameters

info
Input. This input parameter accepts the Ddpi_Info processing object.

in_addr
Input. This accepts the address.

amode
Input. This accepts the address mode.

© Copyright IBM Corp. 2004, 2009

ret_addr
Output. This returns the normalized address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful normalization of `Dwarf_Addr`.

DW_DLV_ERROR
This value is returned if:

- `amode` is unknown
- `out_addr` is NULL

ddpi_addr_offset_normalize operation

This operation adds an offset to an address to correct its alignment for the given address mode.

Prototype

```
int ddpi_addr_offset_normalize(
  Ddpi_Info             info,
  Dwarf_Addr            in_addr,
  Dwarf_Unsigned        in_offset,
  Ddpi_Addr_Mode        amode,
  Dwarf_Addr*           ret_addr,
  Ddpi_Error*           error);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

in_addr
Input. This accepts the address.

in_offset
Input. This accepts the address offset.

amode
Input. This accepts the address mode.

ret_addr
Output. This returns the normalized address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful alignment of `Dwarf_Addr`.

DW_DLV_ERROR
This value is returned if:

- `amode` is unknown
- `out_addr` is NULL

Chapter 6. Ddpi_Elf loading API

The `Ddpi_Elf` loading API loads and relocates the ELF object file.

When a compilation unit (CU) is loaded into memory, the relative (logical) addresses stored within the object file are mapped to physical addresses.

The program analysis application must explicitly load the corresponding ELF object file. At that time, the physical addresses within the ELF object file are relocated in memory to correspond to the physical addresses of the CU information. The `Ddpi_Elf` object holds the information required to relocate these addresses.

For more information about creation and use of `Ddpi_Elf` objects, see *DWARF/ELF Extensions Library Reference, SC09-7655*.

ddpi_elf_load_cu operation

The `ddpi_elf_load_cu` operation loads and relocates the ELF object file.

This operation verifies that the ELF object (inside the `Ddpi_Elf` object) has an MD5 signature that matches the CU that has been loaded into memory. If a match is found, the operation adjusts the relocatable physical addresses of the ELF object to correspond to the physical addresses of the CU that has been loaded into memory.

Prototype

```
int  ddpi_elf_load_cu(
     Ddpi_Info           info,
     Ddpi_Elf            pi_elf,
     Ddpi_Error*         error);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

pi_elf
Input. This accepts the `Ddpi_Elf` object containing the file-specific ELF object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon verification that the ELF object (inside the Ddpi_Elf object) has an MD5 signature that matches the CU that has been loaded in memory.

DW_DLV_ERROR
This value is returned if:

- `pi_elf` is NULL
- `pi_elf` does not have a valid ELF descriptor
- The ELF descriptor contains invalid relocation types
- There is insufficient memory

© Copyright IBM Corp. 2004, 2009

DW_DLV_NO_ENTRY
This value is returned if the MD5 signature does not match the CU that is loaded in memory.

Chapter 7. Ddpi_Info APIs

The Debug Data Program Information library (libddpi) provides a repository for gathering information about a program module. A debugger or other program analysis application can use the repository to collect and query information from the program module.

The Ddpi_Info object is the base object for the application model. Most libddpi operations take the Ddpi_Info object as a parameter, either directly or indirectly. If an object is based on the Ddpi_Info type definition, the information that a program generates about itself as it runs is accessible through that object. For more information about the Ddpi_Info data type, see “Ddpi_Info object” on page 9.

A program analysis application can use a single Ddpi_Info object for all its processing. Some applications may use more than one of these objects. For example, a Ddpi_Info object could be created for each currently active application user.

Each Ddpi_Info object is:

- Created when libddpi is initialized by a ddpi_init call.
- Terminated when libddpi is terminated by a ddpi_finish call.

The ddpi_info operations manage:

- Values that were set wheneve the ddpi_init operation was called
- Values that have been added to the information as a result of creating Ddpi_Space objects

ddpi_info_get_mode operation

This operation is used to determine the processing mode of a Ddpi_Info object.

Prototype

```
int ddpi_info_get_mode(
    Ddpi_Info             info,
    Ddpi_Info_Mode*       ret_mode,
    Ddpi_Error*           error);
```

Parameters

info
Input. This input parameter accepts the Ddpi_Info processing object.

ret_mode
Output. This returns the Ddpi_Info mode.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful determination of the processing mode of a Ddpi_Info object.

DW_DLV_NO_ENTRY
Never returned.

© Copyright IBM Corp. 2004, 2009

DW_DLV_ERROR
This value is returned if:

- `info` is NULL
- `ret_mode` is NULL

ddpi_info_get_user_area operation

This operation returns the address of the user area that was allocated for the given `Ddpi_Info` object by the `ddpi_init` operation.

Prototype

```
int ddpi_info_get_user_area(
  Ddpi_Info             info,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

ret_user_area
Output. This returns the user area of Ddpi_Info.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful return of the address of the user area that was allocated for the given Ddpi_Info object by ddpi_init

DW_DLV_ERROR
This value is returned if info is NULL.

DW_DLV_NO_Entry
This value is returned if the user area length is zero.

ddpi_info_list_space operation

This operation returns a list of the `Ddpi_Space` objects associated with the given `Ddpi_Info` object. The `ddpi_info_list_space` operation sets the returned list to an array of `Ddpi_Space` descriptors, and sets the count of the items in that list to the number of entries in the array. The space list must be freed by the caller, but the individual `Ddpi_Space` objects should not be freed by the caller. These are not copies, but are the actual `Ddpi_Space` objects stored in the `Ddpi_Info` object.

The code to free the list is:

```
ddpi_dealloc(info, *ret_space_list, DDPI_DLA_LIST)
```

Prototype

```
int ddpi_info_list_space(
  Ddpi_Info             info,
  Ddpi_Space**          ret_space_list,
  Dwarf_Signed*         ret_space_cnt,
  Ddpi_Error*           error);
```

Parameters

info
Input. This input parameter accepts the Ddpi_Info processing object.

ret_space_list
Output. This returns a list of Ddpi_Space objects.

ret_space_cnt
Output. This returns the count of entries in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful return of the list of the Ddpi_Space objects associated with the given Ddpi_Info object.

DW_DLV_NO_Entry
This value is returned if the list is empty.

DW_DLV_ERROR
This value is returned if:

- info is NULL
- ret_space_list or ret_space_cnt is NULL
- An allocation error occurs

ddpi_info_list_module operation

This operation returns a list of the Ddpi_Module objects associated with the given minor name of an application-executable module. The ddpi_info_list_module operation sets the returned list to an array of Ddpi_Module descriptors, and sets the count of the items in that list to the number of entries in the array. The module list must be freed by the caller, but the individual Ddpi_Module objects should not be freed by the caller. These are not copies, but are the actual Ddpi_Module objects stored in the Ddpi_Info object.

The code to free the list is:

```
ddpi_dealloc(info, *ret_module_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_info_list_module(
  Ddpi_Info             info,
  char *                module_name,
  Ddpi_Module**         ret_module_list,
  Dwarf_Signed*         ret_module_cnt,
  Ddpi_Error*           error);
```

Parameters

info
Input. This input parameter accepts the Ddpi_Info processing object.

module_name
Input. This accepts the Ddpi_Module file name. Only those modules with a minor name that matches the given name are returned. It is not an error to give module_name a value of NULL. In this case only, modules with no minor name would be returned.

ret_module_list
Output. This returns the list of modules.

ret_module_cnt
Output. This returns the count of entries in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful return of the list of the Ddpi_Module objects associated with the given minor name of an application-executable module.

DW_DLV_ERROR
This value is returned if:

- info is NULL
- `ret_module_list`, `ret_module_cnt` or `module_name` is NULL
- An allocation error occurs

DW_DLV_NO_ENTRY
This value is returned if no module with the given name exists.

ddpi_info_set_dwarf_error_handler operation

The `ddpi_info_set_dwarf_error_handler` operation sets the DWARF error handler and error argument that will be passed to the `dwarf_elf_init()` operation whenever a `Dwarf_Debug` object is created by `libddpi` operations. If the debugger calls any `libdwarf` operation and passes in a NULL `Dwarf_Error` object, this operation will handle any `libdwarf` errors that occur.

Prototype

```
int ddpi_info_set_dwarf_error_handler(
   Ddpi_Info            info,
   Dwarf_Handler        errhand,
   Dwarf_Ptr            errarg,
   Ddpi_Error*          error);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

errhand
Input. This parameter accepts either the error-handling callback function or NULL.

errarg
Input. This parameter accepts either a pointer to additional information or NULL. It can be used to pass extra information to `errhand`. The pointer address, and not its target value, is copied into `libddpi` storage. The pointer can be specified when either the `ddpi_init` or `ddpi_error_set_errarg` operation is called. `errarg` can be queried and changed using `ddpi_error` operations.

error
See “The libddpi error parameter” on page 13.

Returned values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if `info` is NULL or invalid.

ddpi_info_set_dbg_dirs operation

The `ddpi_info_set_dbg_dirs` operation sets or resets the list of directories that will be searched whenever a `.dbg` or `.mdbg` file needs to be opened. The list from any previous calls to this operation will be discarded and replaced. Only UNIX System Services directories can be specified, and either the absolute path or relative path can be used.

Prototype

```
int ddpi_info_set_dbg_dirs(
   Ddpi_Info             info,
   char**                dirs,
   Dwarf_Unsigned        num_dirs,
   Ddpi_Error*           error);
```

Parameters

info
Input. This input parameter accepts the `Ddpi_Info` processing object.

dirs
Input. This accepts the list of directories to be searched for `.dbg` files.

num_dirs
Input. This accepts the number of directories in the list.

error
See “The libddpi error parameter” on page 13.

Returned values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `info` is NULL or invalid.
- `dirs` is NULL.

Chapter 8. Ddpi_Space APIs

After a `Ddpi_Info` object has been created, a `Ddpi_Space` object is created for each address space that is used by the application. The `ddpi` operations use it to address different types of storage without significant code changes. If `libddpi` is used for services other than application modelling, you might still need to create `Ddpi_Space` objects so that other `libddpi` operations can access storage.

To define the storage for read and write operations, use the following two objects:

- "Ddpi_GS_Handler callback function" on page 34
- "Ddpi_SS_Handler object" on page 35

These objects define all user storage. You must specify them when you access or assign a machine address space, a remote address space, or a post-mortem image address space.

For more information about accessing storage, see *z/OS CDA User's Guide*.

Ddpi_Space object

The Ddpi_Space object is an opaque data type that contains information regarding application address spaces.

Type definition

```
typedef struct Ddpi_Space_s*     Ddpi_Space;
```

Ddpi_ASID object

An unsigned integer that identifies the address space.

Type definition

```
typedef Dwarf_Unsigned  Ddpi_ASID;
```

Ddpi_ALET object

The access-list entry type (ALET) object.

Type definition

```
typedef Dwarf_Unsigned  Ddpi_ALET;
```

Ddpi_Space_Type object

Identifies the type of the address space.

Type definition

```
typedef enum Ddpi_Space_Type_s {
   Ddpi_ST_Unknown = 0,
   Ddpi_ST_Code    = 1,
   Ddpi_ST_Data    = 2
 } Ddpi_Space_Type;
```

© Copyright IBM Corp. 2004, 2009

Members

Ddpi_ST_Unknown
If this value is 0, the address space contents are invalid.

Ddpi_ST_Code
If this value is 1, this address space contains code and data.

Ddpi_ST_Data
If this value is 2, this address space contains only data.

Ddpi_Xfer_Status object

Contains the error codes for the `ddpi_get_storage` and `ddpi_set_storage` operations.

Type definition

```
typedef enum Ddpi_Xfer_Status_s {
   Ddpi_XFS_ok             =  0,
   Ddpi_XFS_addr_space     =  1,
   Ddpi_XFS_no_page        =  2,
   Ddpi_XFS_no_page_access =  3,
   Ddpi_XFS_read_only      =  4,
   Ddpi_XFS_addr_bad       =  5,
   Ddpi_XFS_page_bad       =  6,
   Ddpi_XFS_xfer_bad       =  7
 } Ddpi_Xfer_Status;
```

Members

Ddpi_XFS_ok
If the returned value is 0, the transfer was successful.

Ddpi_XFS_addr_space
If the returned value is 1, the address space was either invalid or inaccessible.

Ddpi_XFS_no_page
If the returned value is 2, the page was mapped but unavailable.

Ddpi_XFS_no_page_access
If the returned value is 3, the page was mapped but inaccessible.

Ddpi_XFS_read_only
If the returned value is 4, storage contents could not be updated.

Ddpi_XFS_addr_bad
If the returned value is 5, the address in the space was invalid.

Ddpi_XFS_page_bad
If the returned value is 6, the page in the address space was invalid.

Ddpi_XFS_xfer_bad
If the returned value is 7, there was a problem with the transfer mechanism.

Ddpi_GS_Handler callback function

This is the prototype for a function that a `ddpi` operation can call each time it needs to read from storage.

Type definition

```
typedef int (*Ddpi_GS_Handler) (
   Ddpi_StorageLocn      locn,
   Dwarf_Ptr             buffer,
   Dwarf_Unsigned        read_len,
   Dwarf_Unsigned*       ret_actual_len,
   Ddpi_Xfer_Status*     ret_status);
```

Parameters

locn
Input. This accepts the `Ddpi_StorageLocn` object that describes the location from which data will be read.

buffer
Output. This returns the buffer to which data will be written.

read_len
Input. This accepts the data length to read.

ret_actual_len
Output. This returns the actual data length read.

ret_status
Output. This returns the transfer status.

Ddpi_SS_Handler object

This defines the object that a `ddpi` operation uses whenever it needs to write to storage.

Type definition

```
typedef int (*Ddpi_SS_Handler) (
   Ddpi_StorageLocn      locn,
   Dwarf_Ptr             buffer,
   Dwarf_Unsigned        wrt_len,
   Dwarf_Unsigned*       ret_actual_len,
   Ddpi_Xfer_Status*     ret_status);
```

Members

locn
Input. This accepts the Ddpi_StorageLocn object that describes the location to which data will be written.

buffer
Input. This accepts the buffer that contains the data to be written to the location.

wrt_len
Input. This accepts the data length to read.

ret_actual_len
Output. This returns the actual data length read.

ret_status
Output. This returns the transfer status.

ddpi_space_create operation

The ddpi_space_create operation creates a Ddpi_Space object to represent a program address space and returns a descriptor that acts as a handle for accessing the space. The address space may be either an actual machine or post-mortem image. Alternative Ddpi_Info modes allow a single space to represent an application-executable module or a CU-level object file.

When you call the ddpi_space_create operation, and pass a character string as the parameter name, the operation copies the content of the name parameter and stores it to a copy. After the operation returns the copy, you can deallocate the original name to save storage.

Prototype

```
int ddpi_space_create(
    Ddpi_Info            info,
    Ddpi_Space_Type      type,
    char*                name,
    Ddpi_ASID            asid,
    Ddpi_ALET            alet,
    Dwarf_Addr           limit,
    Ddpi_GS_Handler      gs_handler,
    Ddpi_SS_Handler      ss_handler,
    int                  user_area_len,
    Ddpi_Space*          ret_space,
    Ddpi_Error*          error);
```

Parameters

info
Input. This accepts a Ddpi_Info object.

name
Input. This accepts an optional space name string. The given value can be NULL.

type
Input. This accepts a space type as described above.

asid
Input. This accepts a space ASID ID.

alet
Input. This accepts a space ALET ID.

limit
Input. This accepts the maximum address within space.

gs_handler
Input. This accepts the address of an operation that libddpi uses to read memory. If this operation is not provided, libddpi fails when it tries to read memory. The value can be NULL.

ss_handler
Input. This accepts the address of an operation that libddpi uses to write to memory. If this operation is not provided, libddpi generates an error when it needs to write to memory. The value can be NULL.

user_area_len
Input. This accepts the user-area length.

ret_space
Output. This returns the space object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the descriptor that acts as a handle for accessing the space.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `info` is NULL
- `ret_space` is NULL
- `user_area_len` is less than zero
- An error occurs during memory allocation

ddpi_space_term operation

The `ddpi_space_term` operation releases all internal resources associated with the `space` descriptor, and invalidates `space`.

When you terminate `space`, you also terminate all its children (such as processes, modules, and threads) that have not already been terminated.

Prototype

```
int ddpi_space_term(
  Ddpi_Space            space,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of all internal resources associated with the `space` descriptor.

DW_DLV_NO_ENTRY
Returned if the space is not found in its parent `space` list.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` that owns `space` is NULL
- An error occurs while deallocating memory

ddpi_space_get_owner operation

The ddpi_space_get_owner operation queries the owner of a given Ddpi_Space object and sets it to the returned Ddpi_Info object.

Prototype

```
int ddpi_space_get_owner(
  Ddpi_Space           space,
  Ddpi_Info*           ret_owner,
  Ddpi_Error*          error);
```

Parameters

space
Input. This accepts the Ddpi_Space object.

ret_owner
Output. This returns the Ddpi_Info object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the owner of the returned Ddpi_Info object to the owner of the given Ddpi_Space object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- space is NULL
- Ddpi_Info associated with space is NULL
- ret_owner is NULL

ddpi_space_get_name operation

The ddpi_space_get_name operation returns the name of a given Ddpi_Space object and sets the returned name to a pointer to a null-terminated string of characters.

ddpi_space_get_name returns the actual version of the name, and not a copy. Do not deallocate the returned pointer.

Prototype

```
int ddpi_space_get_name(
  Ddpi_Space              space,
  char**                  ret_name,
  Ddpi_Error*             error);
```

Parameters

space
Input. This accepts the Ddpi_Space object.

ret_name
Output. This returns a pointer to the name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the returned name.

DW_DLV_NO_ENTRY
Returned if the name of the `Ddpi_Space` is NULL.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL
- `ret_name` is NULL

ddpi_space_set_name operation

The `ddpi_space_set_name` operation assigns a new name to a given `Ddpi_Space` object.

`ddpi_space_set_name` can be used to set the name to NULL. `ddpi_space_set_name` copies the given name. The caller may deallocate `new_name` after the call to save memory.

Prototype

```
int ddpi_space_set_name(
  Ddpi_Space              space,
  char*                   new_name,
  Ddpi_Error*             error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

new_name
Input. This accepts the new name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the returned name to the given space object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL
- An error occurs while allocating the memory for the copy of the name

ddpi_space_get_type operation

The ddpi_space_get_type operation returns the type of a given Ddpi_Space object.

Prototype

```
int ddpi_space_get_type(
  Ddpi_Space           space,
  Ddpi_Space_Type*     ret_type,
  Ddpi_Error*          error);
```

Parameters

space
Input. This accepts the Ddpi_Space object.

ret_type
Output. This returns the space type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the type of the given space object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- space is NULL
- The Ddpi_Info object associated with space is NULL
- ret_type is NULL

ddpi_space_get_asid operation

The ddpi_space_get_asid operation returns the address space identifier (ASID) for a given Ddpi_Space object.

Prototype

```
int ddpi_space_get_asid(
  Ddpi_Space              space,
  Ddpi_ASID*              ret_asid,
  Ddpi_Error*             error);
```

Parameters

space
Input. This accepts the Ddpi_Space object.

ret_asid
Output. This returns the space ASID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the ASID of the given space object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL
- `ret_asid` is NULL

ddpi_space_set_asid operation

The `ddpi_space_set_asid` operation assigns a new address space identifier (ASID) to a given `Ddpi_Space` object.

Prototype

```
int ddpi_space_set_asid(
  Ddpi_Space              space,
  Ddpi_ASID               new_asid,
  Ddpi_Error*             error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

new_asid
Input. This accepts the new space ASID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the returned ASID to the given space object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL

ddpi_space_get_alet operation

The `ddpi_space_get_alet` operation returns the access list entry table (ALET) of a given `Ddpi_Space` object.

Prototype

```
int ddpi_space_get_alet(
  Ddpi_Space              space,
  Ddpi_ALET*              ret_alet,
  Ddpi_Error*             error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

ret_alet
Output. This returns the space ALET.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the ALET.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL
- `ret_alet` is NULL

ddpi_space_set_alet operation

The `ddpi_space_set_alet` operation assigns a new ALET to a given `Ddpi_Space` object.

Prototype

```
int ddpi_space_set_alet(
  Ddpi_Space              space,
  Ddpi_ALET               new_alet,
  Ddpi_Error*             error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

new_alet
Input. This accepts the new space ALET specification.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the returned ALET to the given space object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL

ddpi_space_get_limit operation

The ddpi_space_get_limit operation returns the maximum address size for a given Ddpi_Space object.

Prototype

```
int ddpi_space_get_limit(
  Ddpi_Space              space,
  Dwarf_Addr*             ret_limit,
  Ddpi_Error*             error);
```

Parameters

space
Input. This accepts the Ddpi_Space object.

ret_limit
Output. This returns the maximum address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the maximum address size of the queried space object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- space is NULL
- Ddpi_Info associated with space is NULL
- ret_limit is NULL

ddpi_space_set_limit operation

The ddpi_space_set_limit operation assigns a new maximum address to a given Ddpi_Space object.

Prototype

```
int ddpi_space_set_limit(
  Ddpi_Space              space,
  Dwarf_Addr              new_limit,
  Ddpi_Error*             error);
```

Parameters

space
Input. This accepts the Ddpi_Space object.

new_limit
Input. This accepts the maximum address within space.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the returned maximum address size to the given space object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL

ddpi_space_get_user_area operation

The `ddpi_space_get_user_area` operation returns the address of the user area that was allocated for the given `Ddpi_Space` object by `ddpi_init`.

Prototype

```
int ddpi_space_get_user_area(
  Ddpi_Space            space,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

ret_user_area
Output. This returns the user area of the space.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the pointer to the work space allocated for the queried space object.

DW_DLV_NO_ENTRY
Returned if the user area length is zero.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL
- `ret_user_area` is NULL

ddpi_space_list_process operation

The `ddpi_space_list_process` operation returns a list of the `Ddpi_Process` objects and the number of items in that list.

The `ddpi_space_list_process` operation sets the returned list to a pointer, which points to an array of `Ddpi_Process` descriptors, and sets the `ret_process_cnt` to the number of entries in the array.

The caller must free the process list but not the individual Ddpi_Process objects.

These objects are not copies, but are the actual Ddpi_Process objects stored in the Ddpi_Space.

The code to free the list is:

```
rc = ddpi_dealloc(info, *ret_process_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_space_list_process(
  Ddpi_Space            space,
  Ddpi_Process**        ret_process_list,
  Dwarf_Signed*         ret_process_cnt,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the Ddpi_Space object.

ret_process_list
Output. This returns the process list.

ret_process_cnt
Output. This returns the count of entries in the process list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the pointer to the process list.

DW_DLV_NO_ENTRY
Returned if there are no processes in the list.

DW_DLV_ERROR
Returned if:

- space is NULL
- Ddpi_Info associated with space is NULL
- ret_process_list or ret_process_cnt is NULL

ddpi_space_list_hidden_module operation

The ddpi_space_list_hidden_module operation lists the hidden Ddpi_Module objects associated with the given Ddpi_Space object.

The ddpi_space_list_hidden_module operation sets the returned list to an array of Ddpi_Module descriptors, and sets ret_hidden_module_cnt to the number of entries in the array.

The caller must free the module list but not the individual Ddpi_Module objects. These are not copies, but are the actual Ddpi_Module objects stored in the Ddpi_Space.

The code to free the hidden-module list is:

```
rc = ddpi_dealloc(info, *ret_hidden_module_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_space_list_cond(
  Ddpi_Space           space,
  Ddpi_Module**        ret_hidden_module_list,
  Dwarf_Signed*        ret_hidden_module_cnt,
  Ddpi_Error*          error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

ret_hidden_module_list
Output. This returns the list of hidden modules.

ret_hidden_module_cnt
Output. This returns the count of entries in `ret_hidden_module_list`.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the pointer to the list of hidden modules.

DW_DLV_NO_ENTRY
Returned if the list of hidden modules is empty.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL
- `ret_hidden_module_list` or `ret_hidden_module_cnt` is NULL

ddpi_space_unhide_module operation

The `ddpi_space_unhide_module` operation activates the `Ddpi_Module` object in the given `Ddpi_Process`.

The `ddpi_space_unhide_module` operation also moves a module from the hidden-module list of a `Ddpi_Space` object and puts it into a module list of a `Ddpi_Process` object.

Prototype

```
int ddpi_space_unhide_module(
  Ddpi_Space           space,
  Ddpi_Process         process,
  Ddpi_Module          module,
  Ddpi_Error*          error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

process
Input. This accepts the `Ddpi_Process` object.

module
Input. This accepts the `Ddpi_Module` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the module is successfully moved from the list of hidden modules to the module list.

DW_DLV_NO_ENTRY
Returned if the module is not found in the list of hidden modules.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `process` is NULL
- `module` is NULL

ddpi_space_is_hidden_module operation

The `ddpi_space_is_hidden_module` operation queries if the given `Ddpi_Module` object is in the `Ddpi_Space` hidden-module list.

If `ddpi_space_list_class` finds the `Ddpi_Module` object, it assigns a non-zero value to the object. If no `Ddpi_Module` object is found, the `ddpi_space_is_hidden_module` operation assigns the value of the object to zero.

Prototype

```
int ddpi_space_is_hidden_module(
  Ddpi_Space            space,
  Ddpi_Module           module,
  Dwarf_Bool*           ret_bool,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

module
Input. This accepts the `Ddpi_Module` object.

ret_bool
Output. This returns the `Dwarf_Bool` value of zero if `module` is not hidden, and a non-zero value if the `module` is hidden.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when a value has been successfully assigned to the given module.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL
- `ret_bool` is NULL

ddpi_space_delete_module operation

The `ddpi_space_delete_module` operation deletes the given `Ddpi_Module` object in the `Ddpi_Process`.

The `ddpi_space_delete_module` operation removes the `Ddpi_Module` object from the `hidden_module_list`, and it deletes the `Ddpi_Space` object from the `owner_space_list` of the `Ddpi_Module`. If the space is the only reference to the module, the module is terminated and all related storage is released.

Prototype

```
int ddpi_space_delete_module(
  Ddpi_Space            space,
  Ddpi_Module           module,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

module
Input. This accepts the `Ddpi_Module` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when:

- The given module is successfully removed from the appropriate lists
- The module is terminated (if appropriate)
- The associated resources are released (if the module is terminated)

DW_DLV_NO_ENTRY
Returned if the given module is not in the hidden-module list.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL
- `module` is NULL

ddpi_space_find_class operation

The `ddpi_space_find_class` operation returns a `Ddpi_Class` object containing a given address.

Instead of searching through the entire Ddpi_Space until Ddpi_Class is found, the ddpi_space_find_class operation searches only the address-range table of the Ddpi_Space.

Prototype

```
int ddpi_space_find_class(
  Ddpi_Space            space,
  Dwarf_Addr            addr,
  Ddpi_Class*           ret_class,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the Ddpi_Space object.

addr
Input. This accepts the desired address.

ret_class
Output. This returns the Ddpi_Class object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the Ddpi_Class object containing a given address is successfully returned.

DW_DLV_NO_ENTRY
Returned if:

- space does not have any class.
- The input address is not in the range of the space range table.

DW_DLV_ERROR
Returned if:

- space is NULL
- Ddpi_Info associated with space is NULL
- ret_class is NULL

Chapter 9. Ddpi_Process APIs

Like Ddpi_Space, Ddpi_Process is part of the libddpi application model.

Ddpi_Process is intended to be used by the program analysis application to keep track of information about various processes active in a given address space.

Ddpi_PRID object

Contains the DWARF process ID.

Type definition

```
typedef Dwarf_Unsigned  Ddpi_PRID;
```

Ddpi_Process_Type object

Contains the process ID type.

Type definition

```
typedef struct Ddpi_Process_s*    Ddpi_Process;
```

Ddpi_Process object

The Ddpi_Process object is an opaque data type that contains information about a process.

Type definition

```
typedef struct Ddpi_Process_s*    Ddpi_Process;
```

ddpi_process_create operation

The `ddpi_process_create` operation creates a `Ddpi_Process` object to represent a process and returns a descriptor that represents a handle for accessing a process.

When you call the operation, and pass a character string as the parameter name, the operation copies the contents of name. After the operation returns, you can deallocate the original name to re-use the storage.

Prototype

```
int ddpi_process_create(
  Ddpi_Space            space,
  char*                 name,
  Ddpi_Process_Type     type,
  Ddpi_PRID             id,
  int                   user_area_len,
  Ddpi_Process*         ret_process,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object that represents the address space of the owner of this process.

name
Input. This accepts a process name.

© Copyright IBM Corp. 2004, 2009

type
Input. This accepts a process type.

id Input. This accepts a process ID.

user_area_len
Input. This accepts the user-area length.

ret_process
Output. This returns the process object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the process descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `space` is NULL
- `Ddpi_Info` associated with `space` is NULL
- `ret_process` is NULL
- `user_area_len` is less than zero
- An error occurs while allocating memory

ddpi_process_term operation

The `ddpi_process_term` operation releases all internal resources associated with the process descriptor and invalidates `process`.

Termination of a process triggers the termination of any entities owned by this process such as modules, threads or classes.

Prototype

```
int ddpi_process_term(
  Ddpi_Process           process,
  Ddpi_Error*           error);
```

Parameters

process
Input. This accepts the `Ddpi_Process` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful termination of the process.

DW_DLV_NO_ENTRY
Returned if the process is not found in its parent `process` list.

DW_DLV_ERROR
Returned if:

- process is NULL
- Ddpi_Info or the space associated with process is NULL
- An error occurs during termination of child descriptors
- An error occurs during memory deallocation.

ddpi_process_get_owner operation

The ddpi_process_get_owner operation queries the owning Ddpi_Space of a given Ddpi_Process object and assigns it to the given ret_owner parameter.

Prototype

```
int ddpi_process_get_owner(
  Ddpi_Process            process,
  Ddpi_Space*           ret_owner,
  Ddpi_Error*             error);
```

Parameters

process
Input. This accepts the Ddpi_Process object.

ret_owner
Output. This returns the Ddpi_Space object that owns the given process.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of resources associated with the Ddpi_Info descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- process is NULL
- Ddpi_Info associated with process is NULL
- ret_owner is NULL

ddpi_process_get_id operation

The ddpi_process_get_id operation queries the process ID and assigns it to ret_id.

Prototype

```
int ddpi_process_get_id(
  Ddpi_Process            process,
  Ddpi_PRID*              ret_id,
  Ddpi_Error*             error);
```

Parameters

process
Input. This accepts the Ddpi_Process object.

ret_id
Output. This returns the process ID of the specified `Ddpi_Process`.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the process ID.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `process` is NULL
- `Ddpi_Info` associated with `process` is NULL
- `ret_id` is NULL

ddpi_process_set_id operation

The `ddpi_process_set_id` operation assigns a new ID to a given `Ddpi_Process` object.

Prototype

```
int ddpi_process_set_id(
  Ddpi_Process             process,
  Ddpi_PRID               new_id,
  Ddpi_Error*             error);
```

Parameters

process
Input. This accepts the `Ddpi_Process` object.

new_id
Input. This accepts the new process ID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the process ID.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `process` is NULL
- `Ddpi_Info` associated with `process` is NULL

ddpi_process_get_name operation

The `ddpi_process_get_name` operation queries the name of a given `Ddpi_Process` object and sets the returned name to a pointer to a null-terminated string of characters.

Never deallocate the returned pointer because ddpi_process_get_name returns the actual version of the name, and not a copy.

Prototype

```
int ddpi_process_get_name(
  Ddpi_Process             process,
  char**                  ret_name,
  Ddpi_Error*             error);
```

Parameters

process
Input. This accepts the Ddpi_Process object.

ret_name
Output. This returns the name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the process name.

DW_DLV_NO_ENTRY
Returned if name is NULL.

DW_DLV_ERROR
Returned if:

- process is NULL
- Ddpi_Info associated with process is NULL
- ret_name is NULL

ddpi_process_set_name operation

The ddpi_process_set_name operation assigns a new name to a given Ddpi_Process object, which can be used to set the name to NULL.

Because ddpi_process_set_name copies the given name, you can deallocate new_name after the call.

Prototype

```
int ddpi_process_set_name(
  Ddpi_Process            process,
  char*                   new_name,
  Ddpi_Error*             error);
```

Parameters

process
Input. This accepts the Ddpi_Process object.

new_name
Input. This accepts the new process name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the new process name.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `process` is NULL
- `Ddpi_Info` associated with `process` is NULL
- An error occurs while allocating the memory for the copy of the name

ddpi_process_get_TCBaddr operation

The `ddpi_process_get_TCBaddr` operation returns the task control block (TCB) address for a given `Ddpi_Process` object.

Prototype

```
int ddpi_process_get_TCBaddr(
  Ddpi_Process          process,
  Dwarf_Addr*           ret_tcbaddr,
  Ddpi_Error*           error);
```

Parameters

process
Input. This accepts the `Ddpi_Process` object.

ret_tcbaddr
Output. This returns the TCB address of the process.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the TCB address.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `process` is NULL
- `Ddpi_Info` associated with `process` is NULL
- `ret_context` is NULL

ddpi_process_set_TCBaddr operation

The `ddpi_process_set_TCBaddr` operation assigns a new task control block (TCB) address to a given `Ddpi_Process` object.

Prototype

```
int ddpi_process_set_TCBaddr(
  Ddpi_Process           process,
  Dwarf_Addr             tcbaddr,
  Ddpi_Error*            error);
```

Parameters

process
Input. This accepts the `Ddpi_Process` object.

tcbaddr
Input. This accepts the new TCB address of the process.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the new TCB process name.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `process` is NULL
- `Ddpi_Info` associated with `process` is NULL

ddpi_process_get_type operation

The `ddpi_process_get_type` operation returns a type for a given `Ddpi_Process` object.

Prototype

```
int ddpi_process_get_type(
  Ddpi_Process           process,
  Ddpi_Process_Type*     ret_type,
  Ddpi_Error*           error);
```

Parameters

process
Input. This accepts the `Ddpi_Process` object.

ret_type
Output. This returns the process type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the process type.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- process is NULL
- Ddpi_Info associated with process is NULL
- ret_typoe is NULL

ddpi_process_get_user_area operation

The ddpi_process_get_user_area operation returns the user area for the given Ddpi_Process object.

Prototype

```
int ddpi_process_get_user_area(
  Ddpi_Process           process,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

process
Input. This accepts the Ddpi_Process object.

ret_user_area
Output. This returns the user area of the process.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the user area of the process.

DW_DLV_NO_ENTRY
Returned if the user area length is zero.

DW_DLV_ERROR
Returned if:

- process is NULL
- Ddpi_Info associated with process is NULL
- ret_user_area is NULL

ddpi_process_list_module operation

The ddpi_process_list_module operation returns a list of Ddpi_Module objects for a given Ddpi_Process and the number of items in that list.

The module list can be freed by the caller with the following code:

```
rc = ddpi_dealloc(info, *ret_module_list, DDPI_DLA_LIST);
```

Do not free the individual Ddpi_Module objects because they are the actual Ddpi_Module objects stored in the Ddpi_Process.

Prototype

```
int ddpi_process_list_module(
  Ddpi_Process          process,
  Ddpi_Module**         ret_module_list,
  Dwarf_Signed*         ret_module_cnt,
  Ddpi_Error*           error);
```

Parameters

process
Input. This accepts the `Ddpi_Process` object.

ret_module_list
Output. This returns a list of pointers to the `Ddpi_Module`.

ret_module_cnt
Output. This returns the count of entries in the module list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the process list.

DW_DLV_NO_ENTRY
Returned if the process list is empty.

DW_DLV_ERROR
Returned if:

- `process` is NULL
- `Ddpi_Info` associated with `process` is NULL
- `ret_module_list` or `ret_module_cnt` is NULL

ddpi_process_hide_module operation

The `ddpi_process_hide_module` operation hides a `Ddpi_Module` object that is to be reactivated later.

The `ddpi_process_hide_module` operation moves the `Ddpi_Module` object from the `Ddpi_Process` module list to the `Ddpi_Space` hidden module list.

Prototype

```
int ddpi_process_hide_module(
  Ddpi_Process          process,
  Ddpi_Module           module,
  Ddpi_Error*           error);
```

Parameters

process
Input. This accepts the `Ddpi_Process` object.

module
Input. This accepts the `Ddpi_Module` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the module has been successfully moved to the list of hidden modules.

DW_DLV_NO_ENTRY
Returned if the `module` is already hidden in the address space of the `process`.

DW_DLV_ERROR
Returned if:

- `process` is NULL
- `Ddpi_Info` associated with `process` is NULL
- `module` is NULL

ddpi_process_list_class operation

The `ddpi_process_list_class` operation returns a pointer to an array of the `Ddpi_Class` objects for a given `Ddpi_Process` and the number of entries in the array.

This class list can be freed by the caller with the following code:

```
rc = ddpi_dealloc(info, *ret_class_list, DDPI_DLA_LIST);
```

Do not free the individual `Ddpi_Class` objects because they are the actual objects stored in the `Ddpi_Process`.

Prototype

```
int ddpi_process_list_module(
  Ddpi_Process          process,
  Ddpi_Class**         ret_class_list,
  Dwarf_Signed*         ret_class_cnt,
  Ddpi_Error*           error);
```

Parameters

process
Input. This accepts the `Ddpi_Process` object.

ret_class_list
Output. This returns a list of pointers to the `Ddpi_Class`.

ret_class_cnt
Output. This returns the count of entries in the class list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the array pointer and the number of entries in the array.

DW_DLV_NO_ENTRY
Returned if there are no classes.

DW_DLV_ERROR
Returned if:

- `process` is NULL
- `Ddpi_Info` associated with `process` is NULL
- `ret_class_list` or `ret_class_cnt` is NULL

ddpi_process_list_thread operation

The `ddpi_process_list_thread` operation lists the `Ddpi_Thread` objects associated with the given `Ddpi_Process` object.

The ddpi_process_list_thread operation also sets the returned list to a pointer, which points to an array of Ddpi_Thread descriptors and sets the ret_thread_cnt to the number of entries in the array.

The caller must free the thread list but not the individual Ddpi_Thread objects because these are not copies, but are the actual Ddpi_Thread objects stored in the Ddpi_Process.

The code to free the list is:

```
rc = ddpi_dealloc(info, *ret_thread_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_process_list_thread(
  Ddpi_Process          process,
  Ddpi_Thread**         ret_thread_list,
  Dwarf_Signed*         ret_thread_cnt,
  Ddpi_Error*           error);
```

Parameters

process
Input. This accepts the Ddpi_Process object.

ret_thread_list
Output. This returns the list of threads.

ret_thread_cnt
Output. This returns the count of entries in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the thread list and deallocation of storage.

DW_DLV_NO_ENTRY
Returned if there are no threads associated with this process.

DW_DLV_ERROR
Returned if:

- process is NULL
- Ddpi_Info associated with process is NULL
- ret_thread_list or ret_thread_cnt is NULL

Chapter 10. Ddpi_Thread APIs

Ddpi_Thread APIs keep track of POSIX threads used in a given process.

Ddpi_THID object

The thread ID.

Type definition

```
typedef Dwarf_Unsigned  Ddpi_THID;
```

Ddpi_Thread_Type object

The thread type.

Type definition

```
typedef enum Ddpi_Thread_Type_s {
   Ddpi_TT_Unknown = 0,
   Ddpi_TT_POSIX   = 1
} Ddpi_Thread_Type;
```

Members

Ddpi_TT_Unknown
If this value is 0, the thread type is unknown.

Note: Not recommended.

Ddpi_TT_POSIX
If this value is 1, the thread is a POSIX thread.

Ddpi_Thread object

The Ddpi_Thread object is an opaque data type that tracks threads for a given process.

Type definition

```
typedef struct Ddpi_Thread_s*     Ddpi_Thread;
```

ddpi_thread_create operation

The `ddpi_thread_create` operation creates a `Ddpi_Thread` object to represent a POSIX thread and returns a descriptor that represents a handle for accessing the newly created thread.

When you call the `ddpi_thread_create` operation, and pass a character string as the parameter name, the operation copies the content of the parameter name into its own storage. After the operation returns a value, you can deallocate the original name and re-use the storage.

Note: If the thread type is unknown (set to 0), the effect of this operation is undefined.

© Copyright IBM Corp. 2004, 2009

Prototype

```
int ddpi_thread_create(
  Ddpi_Process          process,
  char*                 name,
  Ddpi_Thread_Type      type,
  Ddpi_THID             id,
  int                   user_area_len,
  Dwarf_Addr            tcbaddr,
  Ddpi_Thread*          ret_thread,
  Ddpi_Error*           error);
```

Parameters

process
Input. This accepts the `Ddpi_Process` object.

name
Input. This accepts a thread name.

type
Input. This accepts a thread type.

id Input. This accepts a thread ID.

user_area_len
Input. This accepts the user-area length.

tcbaddr
Input. This accepts task control block address.

ret_thread
Output. This returns the thread object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the thread object descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `process` is NULL
- `Ddpi_Info` associated with `process` is NULL
- `ret_thread` is NULL
- `user_area_len` is less than zero
- An error occurs while allocating memory

ddpi_thread_term operation

The `ddpi_thread_term` operation releases all internal resources associated with the thread descriptor, and invalidates `thread`.

Prototype

```
int ddpi_thread_term(
  Ddpi_Thread           thread,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of all internal resources associated with the thread descriptor.

DW_DLV_NO_ENTRY
Returned if the given thread is not found in the thread list of the owning process.

DW_DLV_ERROR
Returned if:

- `thread` is NULL
- `Ddpi_Info` associated with `thread` is NULL
- An error occurs while terminating child descriptors
- An error occurs while deallocating memory

ddpi_thread_get_owner operation

The `ddpi_thread_get_owner` operation queries the owner of a given `Ddpi_Thread` object and sets the `Ddpi_Thread` object to the returned `Ddpi_Process` object.

Prototype

```
int ddpi_thread_get_owner(
  Ddpi_Thread             thread,
  Ddpi_Process*           ret_process,
  Ddpi_Error*             error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object.

ret_process
Output. This returns the `Ddpi_Process` object that owns the given thread.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the thread object.

DW_DLV_NO_ENTRY
Returned if the given thread is not found in the thread list of the owning process.

DW_DLV_ERROR
Returned if:

- `thread` is NULL
- `Ddpi_Info` associated with `thread` is NULL

- ret_process is NULL

ddpi_thread_get_name operation

The ddpi_thread_get_name operation queries the name of a given Ddpi_Thread object and sets the returned name to a pointer to a null-terminated string of characters.

Do not deallocate the returned pointer because the ddpi_thread_get_name operation returns the actual version of the name, and not a copy.

Prototype

```
int ddpi_thread_get_name(
  Ddpi_Thread             thread,
  char**                  ret_name,
  Ddpi_Error*             error);
```

Parameters

thread
Input. This accepts the Ddpi_Thread object.

ret_name
Output. This returns the name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the pointer is successfully set to the retrieved thread name.

DW_DLV_NO_ENTRY
Returned if the given thread is not found in the thread list of the owning process.

DW_DLV_ERROR
Returned if:

- thread is NULL
- Ddpi_Info associated with thread is NULL
- ret_name is NULL

ddpi_thread_set_name operation

The ddpi_thread_set_name operation assigns a new name to a given Ddpi_Thread object.

The ddpi_thread_set_name operation can be used to set the name to NULL. Because the ddpi_thread_set_name copies the given name, you can deallocate new_name after the call.

Prototype

```
int ddpi_thread_set_name(
  Ddpi_Thread             thread,
  char*                   new_name,
  Ddpi_Error*             error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object.

new_name
Input. This accepts the new name string. This is the address of a NULL-terminated string containing the new name.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the new name.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `thread` is NULL
- `Ddpi_Info` associated with `thread` is NULL
- An error occurs during memory allocation for the copy of the name

ddpi_thread_get_id operation

The `ddpi_thread_get_id` operation returns the ID for the given `Ddpi_Thread` object.

Prototype

```
int ddpi_thread_get_id(
  Ddpi_Thread             thread,
  Ddpi_THID*              ret_id,
  Ddpi_Error*             error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object.

ret_id
Output. This returns the thread ID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the ID for the given `Ddpi_Thread` object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `thread` is NULL
- `Ddpi_Info` associated with `thread` is NULL
- `ret_id` is NULL

ddpi_thread_set_id operation

The ddpi_thread_set_id operation assigns a new ID to a given Ddpi_Thread object.

Prototype

```
int ddpi_thread_set_id(
  Ddpi_Thread             thread,
  Ddpi_THID               new_id,
  Ddpi_Error*             error);
```

Parameters

thread
Input. This accepts the Ddpi_Thread object.

new_id
Input. This accepts the new thread ID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful acceptance of the new thread ID.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- thread is NULL
- The Ddpi_Info object associated with thread is NULL.

ddpi_thread_get_type operation

The ddpi_thread_get_type operation returns the type of the given Ddpi_Thread object.

Prototype

```
int ddpi_thread_get_type(
  Ddpi_Thread           thread,
  Ddpi_Thread_Type*     ret_type,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the Ddpi_Thread object.

ret_type
Output. This returns the thread type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the thread object type.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `thread` is NULL
- `Ddpi_Info` associated with `thread` is NULL
- `ret_type` is NULL

ddpi_thread_get_TCBaddr operation

The `ddpi_thread_get_TCBaddr` operation returns the task control block (TCB) address for a given `Ddpi_Thread` object.

Prototype

```
int ddpi_thread_get_TCBaddr(
  Ddpi_Thread           thread,
  Dwarf_Addr*           ret_tcbaddr,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object.

ret_tcbaddr
Output. This returns the TCB address of the thread.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the TCB address.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `thread` is NULL
- `Ddpi_Info` associated with `thread` is NULL
- `ret_tcbaddr` is NULL

ddpi_thread_get_user_area operation

The `ddpi_thread_get_user_area` operation returns the address of the user area for the given `Ddpi_Thread` object.

Prototype

```
int ddpi_thread_get_user_area(
  Ddpi_Thread           thread,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object.

ret_user_area
Output. This returns a pointer to the user area of the `Ddpi_Thread`.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the user area address.

DW_DLV_NO_ENTRY
Returned if the user area length is zero.

DW_DLV_ERROR
Returned if:

- `thread` is NULL
- `Ddpi_Info` associated with `thread` is NULL
- `ret_user_area` is NULL

ddpi_thread_list_mutex operation

The `ddpi_thread_list_mutex` operation lists the `Ddpi_Mutex` objects associated with the given `Ddpi_Thread` object.

The `ddpi_thread_list_mutex` operation also sets the `ret_mutex_list` to an array of `Ddpi_Mutex` descriptors and sets the `ret_mutex_cnt` to the number of items in `ret_mutex_list`.

The caller must free the mutex list but not the individual `Ddpi_Mutex` objects because these are not copies, but are the actual `Ddpi_Mutex` objects stored in the `Ddpi_Thread`.

The code to free the mutex list is:

```
rc = ddpi_dealloc(info, *ret_mutex_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_thread_list_mutex(
  Ddpi_Thread           thread,
  Ddpi_Mutex**          ret_mutex_list,
  Dwarf_Signed*         ret_mutex_cnt,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object.

ret_mutex_list
Output. This is where the mutex list will be returned.

ret_mutex_cnt
Output. This returns the count of entries in the mutex list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the mutex list.

DW_DLV_NO_ENTRY
Returned if the mutex list is empty.

DW_DLV_ERROR
Returned if:

- `thread` is NULL
- `Ddpi_Info` associated with `thread` is NULL
- `ret_mutex_list` or `ret_mutex_cnt` is NULL

ddpi_thread_list_cond operation

The `ddpi_thread_list_cond` operation lists the `Ddpi_Cond` objects associated with the given `Ddpi_Thread` object.

The `ddpi_thread_list_cond` operation also sets the `ret_cond_list` to an array of `Ddpi_Cond` descriptors and sets `ret_cond_cnt` to the number of items in `ret_cond_list`.

The caller must free the condition-variable list but not the individual `Ddpi_Cond` objects because these are not copies, but are the actual `Ddpi_Cond` objects stored in the `Ddpi_Thread`.

The code to free the list is:

```
rc = ddpi_dealloc(info, *ret_cond_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_thread_list_cond(
  Ddpi_Thread           thread,
  Ddpi_Cond**          ret_cond_list,
  Dwarf_Signed*         ret_cond_cnt,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object.

ret_cond_list
Output. This returns the list of condition variables.

ret_cond_cnt
Output. This returns the count of entries in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the condition-variable list.

DW_DLV_NO_ENTRY
Returned if there are no condition variables.

DW_DLV_ERROR
Returned if:

- `thread` is NULL
- `Ddpi_Info` associated with `thread` is NULL
- `ret_cond_list` or `ret_cond_cnt` is NULL

ddpi_thread_list_lock operation

The `ddpi_thread_list_lock` operation lists the `Ddpi_Lock` objects associated with the given `Ddpi_Thread` object.

The `ddpi_thread_list_lock` operation also sets `ret_lock_list` to an array of `Ddpi_Lock` descriptors and sets `ret_lock_cnt` to the number of items in `ret_lock_list`.

The caller must free the lock list but not the individual `Ddpi_Lock` objects because these are not copies, but are the actual `Ddpi_Lock` objects stored in the `Ddpi_Thread`.

The code to free the list is:

```
rc = ddpi_dealloc(info, *ret_lock_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_thread_list_lock(
  Ddpi_Thread           thread,
  Ddpi_Lock**           ret_lock_list,
  Dwarf_Signed*         ret_lock_cnt,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object.

ret_lock_list
Output. This returns the list of locks.

ret_lock_cnt
Output. This returns the count of entries in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the lock list.

DW_DLV_NO_ENTRY
Returned if there are no locks.

DW_DLV_ERROR
Returned if:

- `thread` is NULL
- The `Ddpi_Info` object associated with `thread` is NULL

- ret_lock_list or ret_lock_cnt is NULL

ddpi_thread_list_class operation

The ddpi_thread_list_class operation lists the Ddpi_Class objects associated with the given Ddpi_Thread object.

The ddpi_thread_list_class operation also sets ret_class_list to an array of Ddpi_Class descriptors and sets ret_class_cnt to the number of items in ret_class_list.

The caller must free the class list but not the individual Ddpi_Class objects because these are not copies, but are the actual Ddpi_Class objects stored in the Ddpi_Thread.

The code to free the list is:

```
rc = ddpi_dealloc(info, *ret_class_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_thread_list_class(
  Ddpi_Thread           thread,
  Ddpi_Class**          ret_class_list,
  Dwarf_Signed*         ret_class_cnt,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the Ddpi_Thread object.

ret_class_list
Output. This returns the list of classes.

ret_class_cnt
Output. This returns the count of entries in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the class list.

DW_DLV_NO_ENTRY
Returned if there are no classes associated with the thread.

DW_DLV_ERROR
Returned if:

- thread is NULL
- Ddpi_Info associated with thread is NULL
- ret_class_list or ret_class_cnt is NULL

Chapter 11. Ddpi_Lock APIs

Read-write locks allow an application to make concurrent reads and exclusive writes to a protected shared resource. Before it can modify a resource, a thread must have an exclusive write lock. The write lock is granted only after all the read locks have been released. The Ddpi_Lock object, which describes a POSIX lock, is used to track internal information generated during these read-write locks. Each `ddpi_lock` operation creates and modifies the Ddpi_Lock object. A program analysis application can use the Ddpi_Lock object to determine which locks are in use for the application instance.

Ddpi_Lock object

The Ddpi_Lock object is an opaque data type that keeps track of the information within the read-write locks for a user application.

Type definition

```
typedef struct Ddpi_Lock_s*       Ddpi_Lock;
```

Ddpi_LOID object

Contains the lock ID.

Type definition

```
typedef Dwarf_Unsigned  Ddpi_LOID;
```

Ddpi_Lock_State object

Defines the CDA lock states.

Type definition

```
typedef enum Ddpi_Lock_State_s {
    Ddpi_Lock_state_unknown  = 0,
    Ddpi_Lock_for_read       = 1,
    Ddpi_Lock_for_write      = 2,
    Ddpi_Lock_wait_for_read  = 3,
    Ddpi_Lock_wait_for_write = 4,
    Ddpi_Lock_unlock         = 5,
    Ddpi_Lock_unwait         = 6,
    Ddpi_Lock_relock         = 7,
    Ddpi_Lock_unrelock       = 8
 }Ddpi_Lock_State;
```

Members

Ddpi_Lock_state_unknown
0

Ddpi_Lock_for_read
1

Ddpi_Lock_for_write
2

Ddpi_Lock_wait_for_read
3

© Copyright IBM Corp. 2004, 2009

Ddpi_Lock_wait_for_write
4

Ddpi_Lock_unlock
5

Ddpi_Lock_unwait
6

Ddpi_Lock_relock
7

Ddpi_Lock_unrelock
8

ddpi_lock_create operation

The `ddpi_lock_create` operation creates a `Ddpi_Lock` object to describe a POSIX lock and returns a `Ddpi_Lock` descriptor.

Prototype

```
int ddpi_lock_create(
  Ddpi_Thread           thread,
  char*                 name,
  Ddpi_Lock_State       state,
  Ddpi_LOID             id,
  int                   user_area_len,
  Ddpi_Lock*            ret_lock,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object, which owns the lock.

name
Input. This accepts a lock name.

state
Input. This accepts a lock state.

id Input. This accepts a lock ID.

user_area_len
Input. This accepts the user-area length.

ret_lock
Output. This returns the `Ddpi_Lock` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the lock descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `thread` is NULL

- Ddpi_Info associated with thread is NULL
- ret_lock is NULL
- user_area_len is less than zero
- An error occurs during memory allocation

ddpi_lock_term operation

The ddpi_lock_term operation releases all internal resources associated with the descriptor lock and invalidates lock.

Prototype

```
int ddpi_lock_term(
  Ddpi_Lock             lock,
  Ddpi_Error*           error);
```

Parameters

lock
Input. This accepts the Ddpi_Lock object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of all internal resources associated with the descriptor lock.

DW_DLV_NO_ENTRY
Returned if the lock is not found in the lock list of its parent threads.

DW_DLV_ERROR
Returned if:

- lock is NULL
- thread or Ddpi_Info associated with lock is NULL
- An error occurs while deallocating memory

ddpi_lock_get_owner operation

The ddpi_lock_get_owner operation queries the owner of a given Ddpi_Lock object and sets the returned Ddpi_Thread object.

Prototype

```
int ddpi_lock_get_owner(
  Ddpi_Lock             lock,
  Ddpi_Thread*          ret_thread,
  Ddpi_Error*           error);
```

Parameters

lock
Input. This accepts the Ddpi_Lock object.

ret_thread
Output. This returns the Ddpi_Thread object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the thread object has been successfully set.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `lock` is NULL
- `Ddpi_Info` associated with `lock` is NULL
- `ret_thread` is NULL

ddpi_lock_get_name operation

The `ddpi_lock_get_name` operation finds the name of a given `Ddpi_lock` object and sets the returned name to a pointer to a null-terminated string of characters.

Never deallocate the returned pointer because `ddpi_lock_get_name` returns the actual version of the name, and not a copy.

Prototype

```
int ddpi_lock_get_name(
  Ddpi_Lock             lock,
  char**                ret_name,
  Ddpi_Error*           error);
```

Parameters

lock
Input. This accepts the `Ddpi_Lock` object.

ret_name
Output. This returns the name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the pointer has been successfully set.

DW_DLV_NO_ENTRY
Returned if `name` is NULL.

DW_DLV_ERROR
Returned if:

- `lock` is NULL
- `Ddpi_Info` associated with `lock` is NULL
- `ret_name` is NULL

ddpi_lock_set_name operation

The `ddpi_lock_set_name` operation assigns a new name to a given `Ddpi_Lock` object.

The ddpi_lock_set_name operation can be used to set the name to NULL. ddpi_lock_set_name copies the given name. The caller may deallocate new_name after the call to save memory.

Prototype

```
int ddpi_lock_set_name(
  Ddpi_Lock             lock,
  char*                 new_name,
  Ddpi_Error*           error);
```

Parameters

lock
Input. This accepts the Ddpi_Lock object.

new_name
Input. This accepts the new name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the lock object has been successfully renamed.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- lock is NULL
- Ddpi_Info associated with lock is NULL
- An error occurs during memory allocation for the copy of the name

ddpi_lock_get_id operation

The ddpi_lock_get_id operation returns the ID for a given Ddpi_lock object.

Prototype

```
int ddpi_lock_get_id(
  Ddpi_Lock             lock,
  Ddpi_LOID*            ret_id,
  Ddpi_Error*           error);
```

Parameters

lock
Input. This accepts the Ddpi_Lock object.

ret_id
Output. This returns the lock ID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the lock ID.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `lock` is NULL
- `Ddpi_Info` associated with `lock` is NULL
- `ret_id` is NULL

ddpi_lock_set_id operation

The `ddpi_lock_set_id` operation assigns a new ID to a given `Ddpi_Lock` object.

Prototype

```
int ddpi_lock_set_id(
  Ddpi_Lock             lock,
  Ddpi_LOID             new_id,
  Ddpi_Error*           error);
```

Parameters

lock
Input. This accepts the `Ddpi_Lock` object.

new_id
Input. This accepts the new lock ID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the new ID to the lock object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `lock` is NULL
- `Ddpi_Info` associated with `lock` is NULL

ddpi_lock_get_state operation

The `ddpi_lock_get_state` operation returns the state for a given `Ddpi_lock` object.

Prototype

```
int ddpi_lock_get_state(
  Ddpi_Lock             lock,
  Ddpi_Lock_State*      ret_state,
  Ddpi_Error*           error);
```

Parameters

lock
Input. This accepts the `Ddpi_Lock` object.

ret_state
Output. This returns the lock state.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the lock object state.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `lock` is NULL
- `Ddpi_Info` associated with `lock` is NULL
- `ret_state` is NULL

ddpi_lock_set_state operation

The `ddpi_lock_set_state` operation assigns a new state to a given `Ddpi_lock` object.

Prototype

```
int ddpi_lock_set_state(
  Ddpi_Lock             lock,
  Ddpi_Lock_State       new_state,
  Ddpi_Error*           error);
```

Parameters

lock
Input. This accepts the `Ddpi_Lock` object.

new_state
Input. This accepts the new lock state.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the lock state has been successfully reset.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `lock` is NULL
- `Ddpi_Info` associated with `lock` is NULL

ddpi_lock_get_user_area operation

The `ddpi_lock_get_user_area` operation finds the user area for a given `Ddpi_Lock` object and sets the start of the returned user area.

Prototype

```
int ddpi_lock_get_user_area(
  Ddpi_Lock             lock,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

lock
Input. This accepts the `Ddpi_Lock` object.

ret_user_area
Output. This returns the lock user area.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful re-initialization of the lock object.

DW_DLV_NO_ENTRY
Returned if the user area length is zero.

DW_DLV_ERROR
Returned if:

- `lock` is NULL
- `Ddpi_Info` associated with `lock` is NULL
- `ret_user_area` is NULL

Chapter 12. Ddpi_Mutex APIs

Mutual exclusion (mutex) locks are used to synchronize thread execution so that only one thread can execute a critical section of code at any one time. The Ddpi_Mutex object keeps track of the information within the mutual exclusion locks that are used within the application.

Ddpi_Mutex object

The Ddpi_Mutex object is an opaque data type that tracks the information within the mutual exclusion locks that are used within the application.

Type definition

```
typedef struct Ddpi_Mutex_s*       Ddpi_Mutex;
```

Ddpi_MUID object

Contains the Mutex ID.

Type definition

```
typedef Dwarf_Unsigned  Ddpi_MUID;
```

Ddpi_Mutex_State object

Contains the mutex state.

Type definition

```
typedef enum Ddpi_Mutex_State_s {
   Ddpi_Mutex_State_unknown = 0,
   Ddpi_Mutex_lock          = 1,
   Ddpi_Mutex_unlock        = 2,
   Ddpi_Mutex_wait          = 3,
   Ddpi_Mutex_unwait        = 4,
   Ddpi_Mutex_relock        = 5,
   Ddpi_Mutex_unrelock      = 6
} Ddpi_Mutex_State;
```

Members

Ddpi_Mutex_state_unknow
0

Ddpi_Mutex_lock
1

Ddpi_Mutex_unlock
2

Ddpi_Mutex_wait
3

Ddpi_Mutex_unwait
4

Ddpi_Mutex_relock
5

Ddpi_Mutex_unrelock
6

© Copyright IBM Corp. 2004, 2009

ddpi_mutex_create operation

The ddpi_mutex_create operation creates a Ddpi_Mutex object to describe a POSIX mutex and assigns a Ddpi_Mutex descriptor.

Prototype

```
int ddpi_mutex_create(
  Ddpi_Thread           thread,
  char*                 name,
  Ddpi_Mutex_State      state,
  Ddpi_MUID             id,
  int                   user_area_len,
  Ddpi_Mutex*           ret_mutex,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the Ddpi_Thread object, which owns the mutex.

name
Input. This accepts a mutex name.

state
Input. This accepts a mutex state.

id Input. This accepts a mutex ID.

user_area_len
Input. This accepts the user-area length.

ret_mutex
Output. This returns the mutex object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the POSIX mutex descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- thread is NULL
- Ddpi_Info associated with thread is NULL
- ret_mutex is NULL
- user_area_len is less than zero
- An error during memory allocation.

ddpi_mutex_term operation

The ddpi_mutex_term operation releases all internal resources associated with the mutex descriptor and invalidates mutex.

Prototype

```
int ddpi_mutex_term(
  Ddpi_Mutex            mutex,
  Ddpi_Error*           error);
```

Parameters

mutex
Input. This accepts the `Ddpi_Mutex` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of all internal resources associated with the mutex descriptor.

DW_DLV_NO_ENTRY
Returned if the mutex is not found in its parent threads `mutex` list.

DW_DLV_ERROR
Returned if:

- `mutex` is NULL
- `Ddpi_Info` or the `thread` associated with `mutex` is NULL
- An error occurs while deallocating memory

ddpi_mutex_get_owner operation

The `ddpi_mutex_get_owner` operation finds the owner of a given `Ddpi_Mutex` object and sets it to the returned `Ddpi_Thread` object.

Prototype

```
int ddpi_mutex_get_owner(
  Ddpi_Mutex             mutex,
  Ddpi_Thread*          ret_thread,
  Ddpi_Error*           error);
```

Parameters

mutex
Input. This accepts the `Ddpi_Mutex` object.

ret_thread
Output. This returns the owner of the mutex.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the returned mutex object value is successfully reset to the thread object value.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- mutex is NULL
- Ddpi_Info associated with mutex is NULL
- ret_thread is NULL

ddpi_mutex_get_name operation

The ddpi_mutex_get_name operation finds the name of a given Ddpi_Mutex object and returns a pointer to the object.

Do not deallocate the pointer.

Prototype

```
int ddpi_mutex_get_name(
  Ddpi_Mutex             mutex,
  char**                 ret_name,
  Ddpi_Error*            error);
```

Parameters

mutex
Input. This accepts the Ddpi_Mutex object.

ret_name
Output. This returns the name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the pointer has been set to the retrieved name.

DW_DLV_NO_ENTRY
Returned if name is NULL.

DW_DLV_ERROR
Returned if:

- mutex is NULL
- Ddpi_Info associated with mutex is NULL
- ret_name is NULL

ddpi_mutex_set_name operation

The ddpi_mutex_set_name operation assigns a new name to a given Ddpi_Mutex object.

The ddpi_mutex_set_name operation can be used to set the name to NULL.

You can deallocate new_name after the call because ddpi_mutex_set_name copies the given name.

Prototype

```
int ddpi_mutex_set_name(
  Ddpi_Mutex             mutex,
  char*                  new_name,
  Ddpi_Error*            error);
```

Parameters

mutex
Input. This accepts the `Ddpi_Mutex` object.

new_name
Input. This accepts the new name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the `mutex` object has been successfully renamed.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `mutex` is NULL
- `Ddpi_Info` associated with `mutex` is NULL
- An error occurs while allocating the memory for the copy of the name

ddpi_mutex_get_id operation

The `ddpi_mutex_get_id` operation returns the ID for a given `Ddpi_Mutex` object.

Prototype

```
int ddpi_mutex_get_id(
  Ddpi_Mutex             mutex,
  Ddpi_MUID*            ret_id,
  Ddpi_Error*           error);
```

Parameters

mutex
Input. This accepts the `Ddpi_Mutex` object.

ret_id
Output. This returns the mutex ID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the `mutex` object ID.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `mutex` is NULL
- `Ddpi_Info` associated with `mutex` is NULL
- `ret_id` is NULL

ddpi_mutex_set_id operation

The ddpi_mutex_set_id operation assigns a new ID to a given Ddpi_Mutex object.

Prototype

```
int ddpi_mutex_set_id(
  Ddpi_Mutex             mutex,
  Ddpi_MUID             new_id,
  Ddpi_Error*           error);
```

Parameters

mutex
Input. This accepts the Ddpi_Mutex object.

new_id
Input. This accepts the new mutex ID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when a new ID has been successfully assigned to mutex.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- mutex is NULL
- Ddpi_Info associated with mutex is NULL

ddpi_mutex_get_state operation

The ddpi_mutex_get_state operation returns the state for a given Ddpi_Mutex object.

Prototype

```
int ddpi_mutex_get_state(
  Ddpi_Mutex             mutex,
  Ddpi_Mutex_State*      ret_state,
  Ddpi_Error*           error);
```

Parameters

mutex
Input. This accepts the Ddpi_Mutex object.

ret_state
Output. This returns the state of the Ddpi_Mutex object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the Ddpi_Mutex object state.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `mutex` is NULL
- The Ddpi_Info object associated with `mutex` is NULL
- `ret_state` is NULL

ddpi_mutex_set_state operation

The `ddpi_mutex_set_state` operation assigns a new state to a given Ddpi_Mutex object.

Prototype

```
int ddpi_mutex_set_state(
  Ddpi_Mutex             mutex,
  Ddpi_Mutex_State       new_state,
  Ddpi_Error*           error);
```

Parameters

mutex
Input. This accepts the Ddpi_Mutex object.

new_state
Input. This accepts the new state of the mutex.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the Ddpi_Mutex object state has been successfully reset.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `mutex` is NULL
- `Ddpi_Info` associated with `mutex` is NULL

ddpi_mutex_get_user_area operation

The `ddpi_mutex_get_user_area` operation finds the user area for a given `Ddpi_Mutex` object and sets the start of the returned user area using the returned value.

Prototype

```
int ddpi_mutex_get_user_area(
  Ddpi_Mutex            mutex,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

mutex
Input. This accepts the `Ddpi_Mutex` object.

ret_user_area
Output. This returns the user area of the mutex.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful re-initialization of the retrieved user area.

DW_DLV_NO_ENTRY
Returned if the user area length is zero.

DW_DLV_ERROR
Returned if:

- `mutex` is NULL
- The Ddpi_Info object associated with `mutex` is NULL
- `ret_user_area` is NULL

Chapter 13. Ddpi_Cond APIs

Condition variables are used to block thread execution until a particular condition is satisfied. The condition is tested under the protection of a mutual exclusion (mutex) lock. The Ddpi_Cond object keeps track of the information within the condition variables that are used within the user application.

Ddpi_Cond object

This represents the state of the current condition variable.

Type definition

```
typedef enum Ddpi_Cond_State_s {
  Ddpi_Cond_State_unknown  = 0,
  Ddpi_Cond_wait           = 1,
  Ddpi_Cond_unwait         = 2
} Ddpi_Cond_State;
```

Members

Ddpi_Cond_state_unknown
If this value is 0, the condition variable state is unknown. It is not recommended that you set the condition state as 0.

Ddpi_Cond_wait
If this value is 1, the condition variable is currently in the wait state.

Ddpi_Cond_unwait
If this value is 2, the condition variable is currently in the unwait state.

Ddpi_CVID object

The CVID data type.

Type definition

```
typedef Dwarf_Unsigned  Ddpi_CVID;
```

Ddpi_Cond object

The Ddpi_Cond opaque data type

- Tracks the information within the condition variables that are used within the user application.
- Contains information regarding application of a condition variable.

Type definition

```
typedef struct Ddpi_Cond_s*       Ddpi_Cond;
```

ddpi_cond_create operation

The `ddpi_cond_create` operation creates a `Ddpi_cond` object to describe a POSIX condition and assigns a `Ddpi_Cond` descriptor.

© Copyright IBM Corp. 2004, 2009

Prototype

```
int ddpi_cond_create(
  Ddpi_Thread           thread,
  char*                 name,
  Ddpi_Cond_State       state,
  Ddpi_CVID             id,
  int                   user_area_len,
  Ddpi_Cond*            ret_cond,
  Ddpi_Error*           error);
```

Parameters

thread
Input. This accepts the `Ddpi_Thread` object, which owns the condition variable.

name
Input. This accepts a condition-variable name.

state
Input. This accepts a condition-variable state.

id Input. This accepts a condition-variable ID or 0.

user_area_len
Input. This accepts the user area length.

ret_cond
Output. This returns the condition-variable object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the POSIX condition descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `thread` or its associated `Ddpi_Info` is NULL.
- `ret_cond` is NULL.
- `user_area_len` is less than zero.
- An error occurs during memory allocation.

ddpi_cond_term operation

The `ddpi_cond_term` operation releases all internal resources associated with the condition-variable descriptor and invalidates `cond`.

Prototype

```
int ddpi_cond_term(
  Ddpi_Cond             cond,
  Ddpi_Error*           error);
```

Parameters

cond
Input. This accepts the `Ddpi_Cond` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of resources associated with the condition-variable descriptor.

DW_DLV_NO_ENTRY
Returned if the condition variable is not found in its parent thread's `cond` list.

DW_DLV_ERROR
Returned if:

- `cond` or its associated thread or `Ddpi_Info` is NULL.
- `ret_cond` is NULL.
- An error occurs during memory deallocation.

ddpi_cond_get_owner operation

The `ddpi_cond_get_owner` operation queries the owner of a given Ddpi_Cond object and sets the returned `Ddpi_Thread` object.

Prototype

```
int ddpi_cond_get_owner(
  Ddpi_Cond             cond,
  Ddpi_Thread*          ret_thread,
  Ddpi_Error*           error);
```

Parameters

cond
Input. This accepts the `Ddpi_Cond` object.

ret_thread
Output. This returns the `Ddpi_Thread` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the condition object has been successfully reset.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `cond` or its associated `Ddpi_Info` object is NULL.
- `ret_thread` is NULL.

ddpi_cond_get_name operation

The `ddpi_cond_get_name` operation queries the name of a given `Ddpi_Cond` object and sets the returned name to a pointer to a null-terminated string of characters.

Never deallocate the returned pointer because ddpi_cond_get_name returns the actual version of the name, and not a copy.

Prototype

```
int ddpi_cond_get_name(
  Ddpi_Cond             cond,
  char**                ret_name,
  Ddpi_Error*           error);
```

Parameters

cond
Input. This accepts the Ddpi_Cond object.

ret_name
Output. This returns the name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the pointer has been successfully set to the retrieved name.

DW_DLV_NO_ENTRY
Returned if name is NULL.

DW_DLV_ERROR
Returned if:

- cond or its associated Ddpi_Info object is NULL.
- ret_name is NULL.

ddpi_cond_set_name operation

The ddpi_cond_set_name operation assigns a new name to a given Ddpi_Cond object.

It can be used to set the name to NULL.

You can deallocate new_name after the call because ddpi_cond_set_name copies the given name.

Prototype

```
int ddpi_cond_set_name(
  Ddpi_Cond             cond,
  char*                 new_name,
  Ddpi_Error*           error);
```

Parameters

cond
Input. This accepts the Ddpi_Cond object.

new_name
Input. This accepts the new name string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the Ddpi_Cond object has been successfully renamed.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- cond or its associated Ddpi_Info object is NULL.
- An error occurs during memory allocation for the copy of the name

ddpi_cond_get_id operation

The ddpi_cond_get_id operation returns the ID for a given Ddpi_Cond object.

Prototype

```
int ddpi_cond_get_id(
  Ddpi_Cond             cond,
  Ddpi_CVID*            ret_id,
  Ddpi_Error*           error);
```

Parameters

cond
Input. This accepts the Ddpi_Cond. object.

ret_id
Output. This returns the conditional variable ID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the conditional variable ID.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- cond or its associated Ddpi_Info object is NULL.
- ret_id is NULL.

ddpi_cond_set_id operation

The ddpi_cond_set_id operation assigns a new ID to a given Ddpi_Cond object.

Prototype

```
int ddpi_cond_set_id(
  Ddpi_Cond             cond,
  Ddpi_CVID             new_id,
  Ddpi_Error*           error);
```

Parameters

cond
Input. This accepts the `Ddpi_Cond` object.

new_id
Input. This accepts the new conditional-variable ID.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the new ID to the `Ddpi_Cond` object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if `cond` or its associated `Ddpi_Info` object is NULL.

ddpi_cond_get_state operation

The `ddpi_cond_get_state` operation returns the state for a given `Ddpi_Cond` object.

Prototype

```
int ddpi_cond_get_state(
  Ddpi_Cond             cond,
  Ddpi_Cond_State*      ret_state,
  Ddpi_Error*           error);
```

Parameters

cond
Input. This accepts the `Ddpi_Cond` object.

ret_state
Output. This returns the state of the conditional variable.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the `Ddpi_Cond` object state.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `cond` or its associated `Ddpi_Info` object is NULL.
- `ret_state` is NULL.

ddpi_cond_set_state operation

The `ddpi_cond_set_state` operation assigns a new state to a given `Ddpi_Cond` object.

Prototype

```
int ddpi_cond_set_state(
  Ddpi_Cond             cond,
  Ddpi_Cond_State       new_state,
  Ddpi_Error*           error);
```

Parameters

cond
Input. This accepts the `Ddpi_Cond` object.

new_state
Input. This accepts the new state of the conditional variable.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of resources associated with the `Ddpi_Info` descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if `cond` or its associated `Ddpi_Info` object is NULL.

ddpi_cond_get_user_area operation

The `ddpi_cond_get_user_area` operation finds the user area for a given `Ddpi_Cond` object and sets the start of the returned user area using the returned value.

Prototype

```
int ddpi_cond_get_user_area(
  Ddpi_Cond             cond,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

cond
Input. This accepts the `Ddpi_Cond` object.

ret_user_area
Output. This returns the user area of the conditional variable.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful re-initialization of the user area.

DW_DLV_NO_ENTRY
Returned if the user area length is zero.

DW_DLV_ERROR
Returned if:

- `cond` or its associated `Ddpi_Info` is NULL

- `ret_user_area` is NULL

Chapter 14. Ddpi_Module APIs

`Ddpi_Module` APIs provide information about the given module.

Use the `ddpi_module` operations to extract:

- The resources allocated to a given module
- The basic format and structure of the module

Ddpi_Module object

The Ddpi_Module object is an opaque data type that contains information about an application-executable module.

Type definition

```
typedef struct Ddpi_Module_s*     Ddpi_Module;
```

Ddpi_Module_Format object

This object contains module object type.

Type definition

```
typedef enum Ddpi_Module_Format_s {
   Ddpi_MT_Unknown     = 0,
   Ddpi_MT_MVS_PDS     = 1,
   Ddpi_MT_MVS_PDSE    = 2,
   Ddpi_MT_MVS_USS_HFS = 3,
   Ddpi_MT_CMS_24      = 4,
   Ddpi_MT_CMS_31      = 5,
   Ddpi_MT_VSE         = 6,
   Ddpi_MT_OBJ_FILE    = 7,
   Ddpi_MT_RAW         = 8
}  Ddpi_Module_Format;
```

Members

Ddpi_MT_Unknown
If this value is 0, the module format is unknown.

Ddpi_MT_MVS_PDS
If this value is 1, the module is a z/OS MVS PDS load module.

Ddpi_MT_MVS_PDSE
If this value is 2, the module is a z/OS MVS PDSE program object.

Ddpi_MT_MVS_USS_HFS
If this value is 3, the module is a z/OS UNIX System Session HFS program object.

Ddpi_MT_CMS_24
If this value is 4, the module is a CMS 24-bit module.

Ddpi_MT_CMS_31
If this value is 5, the module is a CMS 31-bit module.

Ddpi_MT_VSE
If this value is 6, the module is a VSE executable file.

Ddpi_MT_OBJ_FILE
If this value is 7, the module is an object file.

© Copyright IBM Corp. 2004, 2009

Ddpi_MT_RAW
If this value is 8, the module is a raw data storage extent.

Ddpi_Module_Origin object

This object contains the module origin type.

Type definition

```
typedef enum Ddpi_Module_Origin_s {
   Ddpi_MO_Unknown  = 0,
   Ddpi_MO_HFS      = 1,
   Ddpi_MO_MVS_SD   = 2,
   Ddpi_MO_MVS_PD   = 3,
   Ddpi_MO_CMS_DISK = 4,
   Ddpi_MO_CMS_SFS  = 5,
   Ddpi_MO_POSIX    = 6,
   Ddpi_MO_VSE      = 7
 } Ddpi_Module_Origin;
```

Members

Ddpi_MO_Unknown
If this value is 0, the module origin is unknown.

Ddpi_MO_HFS
If this value is 1, the module origin is an HFS path name.

Ddpi_MO_MVS_SD
If this value is 2, the module origin is an MVS sequential dataset.

Ddpi_MO_MVS_PD
If this value is 3, the module origin is an MVS partitioned dataset.

Ddpi_MO_CMS_DISK
If this value is the module origin is a CMS minidisk.

Ddpi_MO_CMS_SFS
If this value is 4, the module origin is a CMS SFS.

Ddpi_MO_POSIX
If this value is 5, the module origin is a CMS POSIX BFS path name.

Ddpi_MO_VSE
If this value is 6, the module origin is a VSE file.

Ddpi_Module_Owner_Type data type

This is the type of module-owner data type.

Type definition

```
typedef enum Ddpi_Module_Owner_Type_s {
   Ddpi_Module_OT_Unknown = 0,
   Ddpi_Module_OT_Space   = 1,
   Ddpi_Module_OT_Process = 2
} Ddpi_Module_Owner_Type;
```

Members

Ddpi_Module_OT_Unknown
If this value is 0, the module-owner data type is unknown.

Ddpi_Module_OT_Space
If this value is 1, the module is currently inactive and is stored in the hidden-module list of the Ddpi_Space object.

Ddpi_Module_OT_Process
If this value is 2, the module is module is currently active and owned by a Ddpi_Process object.

ddpi_module_create operation

The `ddpi_module_create` operation creates a Creates a `Ddpi_Module` object to represent an application-executable module and returns a descriptor that represents a handle for accessing the module.

When you call the `ddpi_module_create` operation, passing character strings as parameter names, the operation copies the content of names and records the address of copies. After this operation returns a value, you can deallocate the original names and save storage.

Prototype

```
int ddpi_module_create(
  Ddpi_Info             info,
  Ddpi_Module_Owner_Type owner_type,
  Ddpi_Module_Owner     owner,
  char*                 major_name,
  char*                 minor_name,
  Ddpi_Module_Format    format,
  Ddpi_Module_Origin    origin,
  int                   user_area_len,
  Ddpi_Module*          ret_module,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts a `Ddpi_Info` object.

owner_type
Input. This accepts the module owner type. This type must be either `Ddpi_Module_OT_Space` or `Ddpi_Module_OT_Process`.

owner
Input. This accepts the module owner.

major_name
Input. This accepts the module location.

minor_name
Input. This accepts the module file name.

format
Input. This accepts the module format.

origin
Input. This accepts the module origin.

user_area_len
Input. This accepts the user-area length.

ret_module
Output. This returns the `Ddpi_Module` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the descriptor that represents a handle for accessing the module.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `info` is NULL.
- `owner` is NULL.
- `owner_type` is neither the `Ddpi_Module_OT_Space` object nor the `Ddpi_Module_OT_Process` object.
- An error occurs during memory allocation.
- `major_name` or `minor_name` was not given.
- `user_area_len` is less than zero.

ddpi_module_term operation

The `ddpi_module_term` operation releases all internal resources associated with the given `Ddpi_Module` object and invalidates `module`.

When you call the `ddpi_module_term` operation, passing character strings as parameter names, the operation copies the content of names and records the address of copies. After this operation returns a value, you can deallocate the original names and save storage.

Prototype

```
int ddpi_module_term(
  Ddpi_Module           module,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of all internal resources associated with the given Ddpi_Module object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `module` is NULL.
- The Ddpi_Info object associated with `module` is NULL.

- A problem occurs during deallocation of the child lists of the Ddpi_Entrypt, Ddpi_Access, or Ddpi_Class object.
- An error occurs during memory deallocation.

ddpi_module_get_access operation

The ddpi_module_get_access operation associates the Ddpi_Access object with the given Ddpi_Module object. Because a Ddpi_Module object can contain a list of Ddpi_Access objects, this operation will return the Ddpi_Access object that was most recently used.

Never deallocate the returned object; ddpi_module_get_access returns the actual Ddpi_Access object, and not a copy.

Prototype

```
int ddpi_module_get_access(
  Ddpi_Module           module,
  Ddpi_Access*          ret_access,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the Ddpi_Module object.

ret_access
Output. This accepts the Ddpi_Access object that was most recently used.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned after the access module is successfully associated with the object module.

DW_DLV_NO_ENTRY
Returned if no Ddpi_Access object from the list was previously used.

DW_DLV_ERROR
This value is returned if:

- module is NULL or invalid.
- The Ddpi_Info object associated with the module is NULL or invalid
- ret_access object is NULL.

ddpi_module_get_owner operation

The ddpi_module_get_owner operation returns the owner of the Ddpi_Module object, which gives instance-specific information.

If you want the list of all owners of all instances of this module, then use ddpi_module_get_all_owners.

Never deallocate the returned owner because ddpi_module_get_owner returns the actual owner, and not a copy.

Prototype

```
int ddpi_module_get_owner(
  Ddpi_Module             module,
  Ddpi_Module_Owner*      ret_owner,
  Ddpi_Module_Owner_Type* ret_owner_type,
  Ddpi_Error*             error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

ret_owner
Output. This returns the owner of the `Ddpi_Module`.

ret_owner_type
Output. This returns the owner type of the `Ddpi_Module`.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of resources associated with the `Ddpi_Info` descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL
- `Ddpi_Info` associated with `module` is NULL
- `ret_owner` is NULL

ddpi_module_list_all_owners operation

The `ddpi_module_list_all_owners` operation queries all owners of the `Ddpi_Module` object and returns a list of owners.

The `ddpi_module_list_all_owners` operation gives instance-specific information for all of the instances of the base-module type. For each item in the owner list, there is a corresponding entry in the owner type list.

Instances occur when `ddpi_module_create` creates two instances of the exact same module at the exact same address. The owner list and the owner type list must be freed by the caller.

Do not free the individual owners.

The code to free the `ret_owner_list` and the `ret_owner_type_list` is as follows:

```
rc = ddpi_dealloc( info, *ret_owner_list, DDPI_DLA_LIST);
rc = ddpi_dealloc( info, *ret_owner_type_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_module_list_all_owners(
  Ddpi_Module           module,
  Ddpi_Module_Owner**   ret_owner_list,
```

```
  Ddpi_Module_Owner_Type**
                      ret_owner_type_list,
  Dwarf_Signed*       ret_owner_count,
  Ddpi_Error*         error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

ret_owner_list
Output. This returns the list of owners.

ret_owner_type_list
Output. This returns the list of owner types corresponding to owner list.

ret_owner_count
Output. This returns the number of items in the `owner_list` and in the `owner_type_list`.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the owner list.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL
- The `Ddpi_Info` object associated with `module` is NULL.
- `ret_owner_list`, `ret_owner_type_list`, or `ret_owner_count` is NULL.
- A problem occurs during memory allocation.

ddpi_module_get_major_name operation

The `ddpi_module_get_major_name` operation finds and returns the major name of the `Ddpi_Module` object.

Never deallocate the returned pointer because this operation returns the actual name, and not a copy.

Prototype

```
int ddpi_module_get_major_name(
  Ddpi_Module         module,
  char**              ret_major_name,
  Ddpi_Error*         error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

ret_major_name
Output. This returns the major name of the module.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the major name of the module has been successfully assigned.

DW_DLV_NO_ENTRY
Returned if the retrieved name is NULL.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL
- The `Ddpi_Info` object associated with `module` is NULL
- `ret_major_name` is NULL
- An internal error occurs

ddpi_module_get_minor_name operation

The `ddpi_module_get_minor_name` operation finds and returns the minor name of the `Ddpi_Module` object.

Never deallocate the returned pointer because this operation returns the actual name, and not a copy.

Prototype

```
int ddpi_module_get_minor_name(
  Ddpi_Module           module,
  char**                ret_minor_name,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

ret_minor_name
Output. This returns the minor name of the module.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the minor name of the object module has been successfully assigned.

DW_DLV_NO_ENTRY
Returned if the retrieved name is NULL.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL
- The Ddpi_Info object associated with `module` is NULL
- `ret_minor_name` is NULL
- An internal error occurs

ddpi_module_get_format operation

The ddpi_module_get_format operation returns the format of the Ddpi_Module object.

Prototype

```
int ddpi_module_get_format(
  Ddpi_Module           module,
  Ddpi_Module_Format*   ret_format,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the Ddpi_Module object.

ret_format
Output. This returns the format of module.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the module format.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- module is NULL
- The Ddpi_Info object associated with module is NULL
- ret_format is NULL

ddpi_module_get_origin operation

The ddpi_module_get_origin operation returns the origin of the Ddpi_Module object.

Prototype

```
int ddpi_module_get_origin(
  Ddpi_Module           module,
  Ddpi_Module_Origin*   ret_origin,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the Ddpi_Module object.

ret_origin
Output. This returns the origin of module.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the module origin.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL
- The `Ddpi_Info` object associated with `module` is NULL
- `ret_origin` is NULL

ddpi_module_get_usage operation

The `ddpi_module_get_usage` operation returns the number of times that the `Ddpi_Module` object was created.

Prototype

```
int ddpi_module_get_usage(
  Ddpi_Module           module,
  Dwarf_Signed*         ret_usage,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the Ddpi_Module object.

ret_usage
Output. This returns the number of times that the Ddpi_Module object was created.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the number of times that the `Ddpi_Module` object was created.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL
- The `Ddpi_Info` object associated with `module` is NULL
- `ret_usage` is NULL

ddpi_module_get_user_area operation

The `ddpi_module_get_user_area` operation finds and returns the user area of the `Ddpi_Module` object.

Prototype

```
int ddpi_module_get_user_area(
  Ddpi_Module           module,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

ret_usage
Output. This returns the user area of `module`.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the user area.

DW_DLV_NO_ENTRY
Returned if the length of the user area is zero.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL
- The `Ddpi_Info` object associated with `module` is NULL
- `ret_usage` is NULL

ddpi_module_list_entrypt operation

The `ddpi_module_list_entrypt` operation lists the `Ddpi_EntryPt` objects associated with the given `Ddpi_Module` object.

The `ddpi_module_list_entrypt` operation also sets:

- `ret_entrypt_list` to an array of Ddpi_EntryPt descriptors
- `ret_entrypt_cnt` of the items in that list to the number of entries in the array

The caller must free the entry-point list but not the individual Ddpi_EntryPt objects because these are not copies, but are the actual `Ddpi_EntryPt` objects stored in the Ddpi_Module.

The code to free `ret_entrypt_list` is:

```
rc = ddpi_dealloc( info, *ret_entrypt_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_module_list_entrypt(
  Ddpi_Module           module,
  Ddpi_EntryPt**        ret_entrypt_list,
  Dwarf_Signed*         ret_entrypt_cnt,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the object.

ret_entrypt_list
Output. This returns the list of the entry point.

ret_entrypt_cnt
Output. This returns the count of the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned after the entry point list is successfully set to an array of `Ddpi_EntryPt` descriptors.

DW_DLV_NO_ENTRY
Returned if the length of the user area is zero.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL
- The `Ddpi_Info` object associated with `module` is NULL
- `ret_entrypt_list` or `ret_entrypt_cnt` is NULL
- A problem during memory allocation.

ddpi_module_list_class operation

The `ddpi_module_list_class` operation lists the `Ddpi_Class` objects in the address-range table of the `Ddpi_Module` object.

The `ddpi_module_list_class` operation also sets:

- `ret_class_list` to an array of `Ddpi_Class` descriptors
- `ret_class_cnt` of the items in that list to the number of entries in the array

The caller must free the entry-point list but not the individual Ddpi_Class objects because these are not copies, but are the actual `Ddpi_Class` objects stored in the `Ddpi_Module`.

The code to free `ret_class_list` is:

```
rc = ddpi_dealloc( info, *ret_class_list, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_module_list_class (
  Ddpi_Module           module,
  Ddpi_Class**          ret_class_list,
  Dwarf_Signed*         ret_class_cnt,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

ret_class_list
Output. This returns the list of the classes.

ret_class_cnt
Output. This returns the count of the list entries.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned after the entry point list is successfully associated with the array of descriptors.

DW_DLV_NO_ENTRY
Returned if the list of entries is empty.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL.
- The `Ddpi_Info` object associated with `module` is NULL.
- `ret_class_list` or `ret_class_cnt` is NULL.
- A problem during memory allocation.

ddpi_module_find_space operation

The `ddpi_module_find_space` operation returns the `Ddpi_Space` object that directly or indirectly owns the given `Ddpi_Module` object.

A `Ddpi_Module` object can be either active or hidden. If it is active, then the `Ddpi_Module` object is owned by a `Ddpi_Process` object. If it is hidden, the `Ddpi_Module` object is on the hidden-module list, and is owned directly by the `Ddpi_Space` object.

Prototype

```
int  ddpi_module_find_space(
      Ddpi_Module         module,
      Ddpi_Space*         ret_space,
      Ddpi_Error*         error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

ret_space
Output. This returns the `Ddpi_Space` object for this instance.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the module owner.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL.
- The Ddpi_Info object associated with `module` is NULL.
- `ret_space` is NULL.

ddpi_module_extract_C_CPP_information operation

The ddpi_module_extract_C_CPP_information operation uses the given Ddpi_Module object to locate the module or program object, and extract all of the information needed to load a DWARF object for C/C++.

The ddpi_module_extract_C_CPP_information operation requires that a Ddpi_EntryPt with the symbol name of CEESTART has been created to properly point to the entry point of the module.

This is a specific helper operation for C/C++ debugging. It does not support other languages. It will not attempt to process a non-C/C++ compile unit (CU).

ddpi_module_extract_C_CPP_information creates and lists the Ddpi_Elf objects associated with the given Ddpi_Module object. It then sets the ret_elf_list to an array of Ddpi_Elf descriptors, and sets the ret_elf_cnt of the items in that list to the number of entries in the array.

The caller must free the Ddpi_Elf list but not the individual Ddpi_Elf objects because these are not copies, but are the actual Ddpi_Elf objects stored in the Ddpi_Module.

The code to free ret_elf_list is:

```
rc = ddpi_dealloc( info, *ret_elf_list, DDPI_DLA_LIST);
```

Prototype

```
int  ddpi_module_extract_C_CPP_information (
       Ddpi_Module         module,
       Ddpi_Elf**          ret_elf_list,
       int*                ret_elf_cnt,
       Ddpi_Error*         error);
```

Parameters

module
Input. This accepts the Ddpi_Module object.

ret_elf_list
Output. This returns list of Ddpi_Elf objects.

ret_elf_cnt
Output. This returns count of the list entries.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned after the ELF list is successfully associated with an array of ELF descriptors.

DW_DLV_NO_ENTRY
Returned if no information could be extracted.

DW_DLV_ERROR
This value is returned if:

- module is NULL or invalid.

- The Ddpi_Info object associated with the module is either NULL or invalid.
- ret_elf_list or ret_elf_cnt is NULL.
- There are no entry points on module.
- An entry point named CEESTART is not found.
- The given entry point does not have a storage extent attached (class).
- An error occurs finding the low address.

ddpi_module_extract_debug_info operation

The ddpi_module_extract_debug_info operation extracts, from the module map, all the information needed to create the DWARF debugging information. The extracted information is used to create ELF objects, as needed. The ELF objects are used to create the DIEs used by the debugger. A separate Ddpi_Access object will be created to own each Ddpi_Elf object. The list of Ddpi_Access objects created will be owned by the given Ddpi_Module object. Note that, if the program analysis application is using the module map, it needs to use this operation instead of the ddpi_module_extract_C_CPP_information() operation.

Successful implementation of the ddpi_module_extract_debug_info operation depends on the following conditions:

- The module has been set up with the correct address range.
- The program analysis application provides functions that retrieve information from the module map.

Prototype

```
int ddpi_module_extract_debug_info(
  Ddpi_Module          module,
  int                  elf_user_area_len,
  Dwarf_Bool*          ret_mod_map,
  Ddpi_Error*          error);
```

Parameters

module
Input. This accepts the given Ddpi_Module object.

elf_user_area_len
Input. This accepts the user area length required for any Ddpi_Elf objects that might be created.

ret_mod_map
Output. This is returned whether or not a module map was found for the module.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
No information could be extracted.

DW_DLV_ERROR
This value is returned if:

- The module value is NULL or invalid.

- The Ddpi_Info object associated with the module is NULL or invalid
- ret_mod_map is NULL.
- An error occurs during the search for the low-order address.
- An error occurs during creation of a Ddpi_Access or Ddpi_Elf object.
- An error occurs during memory allocation.

ddpi_module_find_wsa operation

The ddpi_module_find_wsa operation finds the writable static area (WSA) for a given Ddpi_Module object.

ddpi_module_find_wsa can be called only after the module has been entered because it needs valid stack and registers for the given object.

This operation may be invoked by the expression evaluator if a WSA is needed in the given expression.

Prototype

```
int  ddpi_module_find_wsa(
            Ddpi_Info           info,
            Ddpi_Module         module,
            Ddpi_MachineState   machinestate,
            Ddpi_StackState     stackstate,
            Ddpi_Class*         ret_wsa_class,
            Ddpi_Error*         error);
```

Parameters

info
Input. This accepts the libddpi consumer object.

module
Input. This accepts the Ddpi_Module object associated with the WSA.

machinestate
Input. This accepts the machine state of a live function in the module.

stackstate
Input. This accepts the stack state of the same live function in the module for the machine state.

ret_wsa_class
Output. This returns the WSA class.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the WSA class.

DW_DLV_NO_ENTRY
Returned if no WSA was found.

DW_DLV_ERROR
This value is returned if:

- module is NULL or invalid.
- The Ddpi_Info object associated with the module is either NULL or invalid.

- ret_wsa_class is NULL.
- machinestate or stackstate is NULL. If the WSA class is not already defined on module, then these parameters will be used to find and create a WSA class on module.
- The Language Environment (LE) is unable to find the WSA.

ddpi_module_get_dwarf_error operation

The ddpi_module_get_dwarf_error operation returns a pointer to the Dwarf_Error object for the given Ddpi_Module.

The Dwarf_Error object is found within the Ddpi_Access object, which is owned by the Ddpi_Module object.

Use the ddpi_module_get_dwarf_error operation if your application employs both a Ddpi_Error object and a Dwarf_Error object. It can extract the Dwarf_Error pointer and pass it to all libdwarf calls.

You must terminate the owning libdwarf instance to deallocate Dwarf_Error when ddpi_finish is called.

Prototype

```
int  ddpi_module_get_dwarf_error(
       Ddpi_Module         module,
       Dwarf_Error**       ret_dwarf_error,
       Ddpi_Error*         error);
```

Parameters

module
Input. This accepts the Ddpi_Module object.

ret_dwarf_error
Output. This returns the Dwarf_Error object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the pointer to the DWARF error module.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- module is NULL.
- ret_dwarf_error is NULL
- An error occurs during memory allocation.

ddpi_module_list_function operation

The ddpi_module_list_function operation retrieves a list of all the functions with the given name. This list will include all the functions whose fully qualified name (which is prefixed with a C++ class name, if applicable) or unqualified name matches the given name. The given name could also be a portion of the fully

qualified name. For example, the given name could be `Classname::function`, without the function parameters. If the given name is NULL, all of the functions in the module will be returned. The returned list will contain `Ddpi_Function` objects sorted by unqualified name.

The calling function should deallocate the returned list by using the following operation:

```
ddpi_dealloc(info, *ret_functions, DDPI_DLA_LIST);
```

Note: The actual `Ddpi_Function` objects must not be deleted. They will be deleted when the `Ddpi_Module` object is deleted.

Prototype

```
int ddpi_module_list_function(
   Ddpi_Module          module,
   char*                name,
   Ddpi_Function**      ret_functions,
   Dwarf_Unsigned*      ret_count,
   Ddpi_Error*          error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

name
Input. This accepts the name of a function. If NULL is accepted, a list of all of the functions in the module will be returned.

ret_functions
Output. This returns a list of `Ddpi_Function` objects with the given name.

ret_count
Output. This returns the number of `Ddpi_Function` objects returned.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
The function was not found in the module, or a module map hasn't been created for the module.

DW_DLV_ERROR
This value is returned if:

- `module` is NULL or invalid.
- The `Ddpi_Info` object associated with the module is either NULL or invalid.
- `ret_functions` is NULL.
- `ret_count` is NULL.
- An error occurs during memory allocation.

ddpi_module_list_variable operation

The ddpi_module_list_variable operation rtrieves a list of all the global variables with the given name. This list will include all the variables whose fully qualified name (which is prefixed with a C++ class name, if applicable) or unqualified name matches the given name. If the given name is NULL, all of the global variables in the module will be returned. The returned list will contain Ddpi_Variable objects sorted by unqualified name.

The calling function should deallocate the returned list by using the following operation:

```
ddpi_dealloc(info, *ret_variables, DDPI_DLA_LIST);
```

Note: The actual Ddpi_Variable objects must not be deleted. They will be deleted when the Ddpi_Module is deleted.

Prototype

```
int ddpi_module_list_variable(
   Ddpi_Module           module,
   char*                 name,
   Ddpi_Variable**       ret_variables,
   Dwarf_Unsigned*       ret_count,
   Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the Ddpi_Module object.

name
Input. This accepts the name of an external type. If NULL is accepted, a list of all of the external types in the module will be returned.

ret_variables
Output. This returns a list of Ddpi_Variable objects with the given name.

ret_count
Output. This returns the number of Ddpi_Variable objects returned.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
Returned if the variable was not found in the module, or a module map hasn’t been created for the module.

DW_DLV_ERROR
This value is returned if:

- module is NULL or invalid.
- The Ddpi_Info object associated with the module is either NULL or invalid.
- ret_variables is NULL.
- ret_count is NULL.
- An error occurs during memory allocation.

ddpi_module_list_type operation

The ddpi_module_list_type operation retrieves a list of all the external types with the given name. This list will include all the types whose fully qualified name (which is prefixed with a C++ class name, if applicable) or unqualified name matches the given name. If the given name is NULL, all of the external types in the module will be returned. The returned list will contain Ddpi_Type objects sorted by unqualified name.

The calling function should deallocate the returned list by using the following operation:

```
ddpi_dealloc(info, *ret_types, DDPI_DLA_LIST);
```

Note: The individual Ddpi_Type objects must not be deleted. They will be deleted when the module is deleted.

Prototype

```
int ddpi_module_list_type(
   Ddpi_Module           module,
   char*                 name,
   Ddpi_Type**           ret_types,
   Dwarf_Unsigned*       ret_count,
   Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the Ddpi_Module object.

name
Input. This accepts the external type name, or NULL to return all external types.

ret_types
Output. This returns a list of Ddpi_Type objects with the given name.

ret_count
Output. This returns the number of Ddpi_Type objects returned.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is returned if the type was not found in the module, or a module map hasn’t been created for the module

DW_DLV_ERROR
This value is returned if:

- module is NULL or invalid.
- The Ddpi_Info object associated with the module is either NULL or invalid.
- ret_types is NULL.
- ret_count is NULL.
- An error occurs during memory allocation.

ddpi_module_list_sourcefile operation

The ddpi_module_list_sourcefile operation returns a list of Ddpi_Sourcefile objects whose names match the given name. All Ddpi_Sourcefile objects with that file name in any path will be returned. If the given name is NULL, all of the source files in the module will be returned. Note that if the same source file is used in more than one compilation unit, there will be a separate Ddpi_Sourcefile object for each compilation unit. The Ddpi_Sourcefile objects will be sorted by the file name without the full path.

The calling function should deallocate the returned list by using the following operation:

```
ddpi_dealloc(info, *ret_sourcefiles, DDPI_DLA_LIST);
```

Note: The individual Ddpi_Sourcefile objects in the list must not be deleted. They will be deleted when the module is deleted.

Prototype

```
int ddpi_module_list_sourcefile(
   Ddpi_Module           module,
   char*                 name,
   Ddpi_Sourcefile**     ret_sourcefiles,
   Dwarf_Unsigned*       ret_count,
   Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the Ddpi_Module object.

name
Input. This accepts the source file name, or NULL to return all source files. The name can be a full path name, a file name without a path, a PDS name, or a sequential dataset name.

ret_sourcefiles
Output. This returns the list of Ddpi_Sourcefile objects in the module with the given source file name.

ret_count
Output. This returns the number of Ddpi_Sourcefile objects in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
The given source file name is not represented by any of the Ddpi_Sourcefile objects in the module, or a module map hasn't been created for the module.

DW_DLV_ERROR
This value is returned if:

- module is NULL or invalid.
- The Ddpi_Info object associated with the module is either NULL or invalid.
- ret_sourcefiles is NULL.

- ret_count is NULL.
- An error occurs while allocating memory.

ddpi_module_list_elf operation

Lists all of the Ddpi_Elf objects associated with the given Ddpi_Module object. It sets ret_elfs to an array of Ddpi_Elf objects and sets ret_elf_count to the number of entries in the array. The returned list will be sorted in ascending order by CU address.

The calling function should deallocate the returned list by using the following operation:

```
ddpi_dealloc(info, *ret_elfs, DDPI_DLA_LIST);
```

Note: The actual Ddpi_Elf objects must not be deallocated, as these are the actual Ddpi_Elf objects from the Ddpi_Access objects.

Prototype

```
int ddpi_module_list_elf(
   Ddpi_Module           module,
   Ddpi_Elf**            ret_elfs,
   Dwarf_Unsigned*       ret_elf_count,
   Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the Ddpi_Module object.

ret_elfs
Output. This returns the list of Ddpi_Elf objects in the module.

ret_elf_count
Output. This returns the number of Ddpi_Elf objects in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
There are no Ddpi_Elf objects associated with the module.

DW_DLV_ERROR
This value is returned if:

- module is NULL or invalid.
- ret_elfs is NULL.
- ret_elf_count is NULL.

ddpi_module_find_elf_given_address operation

The ddpi_module_find_elf_given_address operation returns the Ddpi_Elf object that corresponds to the address range in the loaded module that contains the given address. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_module_find_elf_given_address(
   Ddpi_Module           module,
   Dwarf_Addr            address,
   Ddpi_Elf*             ret_elf,
   Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

address
Input. This accepts an address in the loaded module.

ret_elf
Output. This returns the `Ddpi_Elf` object whose address range contains the given address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
The given address is not within the address range for any of the `Ddpi_Elf` objects in the module

DW_DLV_ERROR
This value is returned if:

- `module` is NULL or invalid.
- `ret_elf` is NULL.

Chapter 15. Ddpi_Access APIs

The Ddpi_Access object contains and controls the items necessary to load and use ELF and DWARF objects.

When an error condition is triggered, and the resulting `Ddpi_Error` object indicates the error occurred during a call to a `libdwarf` operation, a program analysis application can extract further information by querying the `Dwarf_Error` object, which is stored in the `Ddpi_Access` object.

That `Ddpi_Access` object contains:

- The `libdwarf` consumer object (`Dwarf_Debug`). Each `libddpi` instance has a single `Dwarf_Debug` object, which can work with one or more `Ddpi_Elf` objects.
- A `Dwarf_Error` object, which is used for all calls to `libdwarf` operations. For example, if DDPI_DLE_DWARF_ERROR (error code 138) is returned, the `Dwarf_Error` object can be queried to determine the cause and type of the error.
- A list of `Ddpi_Elf` objects. It is recommended that a `Ddpi_Elf` object exist for each compilation unit (CU), debug-information file (debugging unit), or section of a unit. Each `Ddpi_Elf` object contains the handle to the ELF object file.

To access the `Dwarf_Error` object, you can use either of the following operations:

- “ddpi_access_get_dwarf_error operation” on page 129
- “ddpi_module_get_dwarf_error operation” on page 115

Both of these operations return the same Dwarf_Error object.

To create the Ddpi_Elf objects, you can use either of the following operations:

- “ddpi_module_extract_C_CPP_information operation” on page 112
- “ddpi_elf_create operation” on page 133

Ddpi_Access object

The Ddpi_Access object contains and controls the items needed to access or load ELF and DWARF objects. It has an opaque data type.

Type definition

```
typedef struct Ddpi_Access_s*      Ddpi_Access;
```

ddpi_access_create operation

The `ddpi_access_create` operation creates a `Ddpi_Access` object and associates it with the `Ddpi_Module`.

The `Ddpi_Access` object manages access to the debugging data in an ELF file.

Note: Because no verification is done on input parameters, you can update the fields with the `ddpi_access_set_*` operations.

© Copyright IBM Corp. 2004, 2009

Prototype

```
int ddpi_access_create(
  Ddpi_Module           module,
  Dwarf_Debug           debug,
  int                   user_area_len,
  Ddpi_Access*          ret_access,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the owning `Ddpi_Module` object.

debug
Input. This accepts an instance of the debug access.

user_area_len
Input. This accepts the length of the user area.

ret_access
Output. This returns the `Ddpi_Access` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful creation of the `Ddpi_Access` object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `module` or `ret_access` variable is NULL.
- `user_area_len` is less than zero.
- An error during memory allocation.

ddpi_access_term operation

The `ddpi_access_term` operation terminates a given Ddpi_Access object and calls `ddpi_elf_term` to terminate `Ddpi_Elf` objects referred to in the `Ddpi_Access` object.

If `ddpi_elf_term` gives an unexpected return code, the return code and error will be returned to the caller of `ddpi_access_term`.

Note: If an error occurs in `ddpi_elf_term`, the remaining portions of the `Ddpi_Elf` list and the `Ddpi_Access` object are not deleted so that `ddpi_access_term` can be called again if the error is resolved. The `ddpi_access_term` operation does not call `libdwarf` to terminate the `Dwarf_Debug` object.

Prototype

```
int ddpi_access_term(
  Ddpi_Access           access,
  Ddpi_Error*           error);
```

Parameters

access
Input. This accepts the `Ddpi_Access` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful termination of the `Ddpi_Access` object.

DW_DLV_NO_ENTRY
Returned if `access` does not match the access referred to in its parent.

DW_DLV_ERROR
This value is returned if:

- `access` or its associated information is NULL
- An error occurs while terminating child descriptors
- An error during memory allocation.

ddpi_access_get_owner operation

The `ddpi_access_get_owner` operation returns the owner of the `Ddpi_Access` object.

Prototype

```
int  ddpi_access_get_owner(
       Ddpi_Access         access,
       Ddpi_Module*        ret_owner,
       Ddpi_Error*         error);
```

Parameters

access
Input. This accepts the Ddpi_Access object.

ret_owner
Output. This returns the Ddpi_Process object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the `Ddpi_Access` object owner.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `access` is NULL.
- The `Ddpi_Access` or `Ddpi_Info` object associated with `access` is NULL.
- `ret_owner` is NULL.

ddpi_access_get_debug operation

The ddpi_access_get_debug operation returns the Dwarf_Debug object that is stored in the given Ddpi_Access object. If the extraction function ddpi_module_extract_debug_info() was used, this operation will open the .dbg file and create a Dwarf_Debug object for the instance, if one does not already exist.

Use only libdwarf operations to modify the given Dwarf_Debug object because ddpi_access_get_debug does not return a copy, but the actual Dwarf_Debug object.

If the Dwarf_Debug object is deleted, use ddpi_access_set_debug to set the Dwarf_Debug pointer in the Ddpi_Access object to NULL.

Prototype

```
int ddpi_access_get_debug(
  Ddpi_Access           access,
  Dwarf_Debug*          ret_debug,
  Ddpi_Error*           error);
```

Parameters

access
Input. This accepts the Ddpi_Access object.

ret_debug
Output. This returns the Dwarf_Debug object.

error
See “The libddpi error parameter” on page 13.

Returned values

DW_DLV_OK
This value is returned upon successful return of the Dwarf_Debug object.

DW_DLV_NO_ENTRY
This value is returned if no valid .dbg files were located.

DW_DLV_ERROR
This value is returned if:

- access is NULL.
- ret_debug is NULL.
- A DWARF error occurs during creating the Dwarf_Debug object for the instance.
- An error occurs during memory allocation.

ddpi_access_set_debug operation

The ddpi_access_set_debug operation sets the Dwarf_Debug object for the given Ddpi_Access object.

The given debug value is not verified in any way. If an older value is being replaced, the caller must properly handle it (for example, the caller must terminate the older value).

Prototype

```
int ddpi_access_set_debug(
  Ddpi_Access           access,
  Dwarf_Debug           debug,
  Ddpi_Error*           error);
```

Parameters

access
Input. This accepts the Ddpi_Access object.

debug
Input. This accepts the debug instance.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the `Dwarf_Debug` object to the given `Ddpi_Access` object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
Returned if:

- `access` is NULL.
- The `Ddpi_Info` object associated with `access` is NULL.

ddpi_access_list_elf operation

The `ddpi_access_list_elf` operation lists the `Ddpi_Elf` objects associated with the given `Ddpi_Access` object.

The `ddpi_access_list_elf` operation also sets the `ret_elfs` field to an array of `Ddpi_Elf` descriptors and sets the `ret_elf_count` of the items in that list to the number of entries in the array.

The caller must free the access list but not the individual `Ddpi_Elf` objects because these are not copies, but are the actual `Ddpi_Elf` objects stored in the `Ddpi_Access` object.

The `Ddpi_Elf` list is accurate only after another `Ddpi_Elf` object is added to, or terminated from, the given `Ddpi_Access` object.

The code to free `ret_elfs` is:

```
ddpi_dealloc(info, *ret_elfs, DDPI_DLA_LIST)
```

Prototype

```
int  ddpi_access_list_elf(
     Ddpi_Access         access,
     Ddpi_Elf**          ret_elfs,
     Dwarf_Unsigned*     ret_elf_count,
     Ddpi_Error*         error);
```

Parameters

access
Input. This accepts the `Ddpi_Access` object.

ret_elfs
Output. This returns the `Ddpi_Elf` list.

ret_elf_count
Output. This returns the `Ddpi_Elf` count.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of resources associated with the `Ddpi_Info` descriptor.

DW_DLV_NO_ENTRY
Returned if there are no entries on the access list.

DW_DLV_ERROR
Returned if:

- `access` or its associated `Ddpi_Info` is NULL.
- `ret_elfs` or `ret_elf_count` is NULL.
- An error occurs during allocation of the list to be returned.

ddpi_access_get_user_area operation

The `ddpi_access_get_user_area` operation returns the user area for the given `Ddpi_Access` object.

Prototype

```
int ddpi_access_get_user_area(
  Ddpi_Access           access,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

access
Input. This accepts the Ddpi_Access object.

ret_user_area
Output. This returns the user area.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the user area.

DW_DLV_NO_ENTRY
Returned if the user-area length is zero.

DW_DLV_ERROR
Returned if:

- `access` or its associated Ddpi_Info object is NULL.

- `ret_elfs` or `ret_elf_count` is NULL.
- An error occurs during allocation of the list to be returned

ddpi_access_get_dwarf_error operation

The `ddpi_access_get_dwarf_error` operation returns the `Dwarf_Debug` object that is stored in the given `Ddpi_Access` object. If the extraction function `ddpi_module_extract_debug_info()` was used, this operation will open the `.dbg` file and create a `Dwarf_Debug` object for the instance, if one does not already exist.

Note: A `Ddpi_Access` object can have a `Dwarf_Error` object.

Use this operation when your application employs both a `Ddpi_Error` object and a `Dwarf_Error` object. It can extract a pointer to the `Dwarf_Error` object and pass it to all `libdwarf` calls.

The `Dwarf_Error` object is not removed from the `Ddpi_Access` object. You must terminate the owning `libdwarf` instance in order to terminate the `Dwarf_Error` object because the `Dwarf_Error` object is not deallocated when `ddpi_finish` is called.

Prototype

```
int  ddpi_access_get_dwarf_error(
       Ddpi_Access         access,
       Dwarf_Error**       ret_dwarf_error,
       Ddpi_Error*         error);
```

Parameters

access
Input. This accepts the `Ddpi_Access` object.

ret_dwarf_error
Output. This returns the `Dwarf_Error` object.

error
See “The libddpi error parameter” on page 13.

Returned values

DW_DLV_OK
This value is returned upon successful return of the DWARF error object.

DW_DLV_NO_ENTRY
This value is returned if no valid `.dbg` files were located.

DW_DLV_ERROR
This value is returned if:

- `access` is NULL.
- `ret_dwarf_error` is NULL.
- A DWARF error occurs during creating the Dwarf_Debug object for the instance.
- An error occurs while allocating memory.

Chapter 16. Ddpi_Elf APIs

This information describes the operations that get and set information in the `Ddpi_Elf` object.

The `Ddpi_Elf` object stores the information needed to access, load, or relocate an ELF object file. For example, it contains a file handle and the address range of the compilation unit (CU) loaded in memory. A `Ddpi_Elf` object is created for each ELF object file. For information on loading and relocating the `Ddpi_Elf` object, see Chapter 6, “Ddpi_Elf loading API,” on page 25.

The program analysis application uses the `Ddpi_Elf` object in order to specify a particular CU or section. Accordingly, the `Ddpi_Elf` object can be set either during creation or by individual routines. For more information on using a `Ddpi_Elf` object , see *Common Debug Architecture User's Guide, SC09-7653*.

For C and C++ modules, you can use the `ddpi_module_extract_C_CPP_information` operation to automatically create `Ddpi_Elf` objects. For more information, see “ddpi_module_extract_C_CPP_information operation” on page 112.

Ddpi_Elf object

A `Ddpi_Elf` object is created for each ELF object file. This structure is used to coordinate access to an ELF object file that has been loaded and pointed to by an ELF descriptor. The data type is opaque.

The `Ddpi_Elf` object holds all information required to relocate the ELF object file:

- The name of the ELF object file
- The location of the PPA2 addresses
- A list of function addresses contained within the CU
- A list of PPA1 addresses contained within the CU
- The MD5 signature
- The `Ddpi_Elf_Source` object
- The high and low memory addresses for the CU that is in memory

To create the `Ddpi_Elf` object, two operations are provided:

- “ddpi_module_extract_C_CPP_information operation” on page 112
- “ddpi_elf_create operation” on page 133

Type definition

```
typedef struct Ddpi_Elf_s*        Ddpi_Elf;
```

Ddpi_Elf_Source object

An ELF object is loaded from an ELF source. The ELF source must remain open as long as the information from it is potentially in use. The Ddpi_Elf_Source object tracks this ELF source.

Ddpi_Elf_Source object has three members:

- A POSIX file (if given a POSIX file descriptor).
- An ANSI C file pointer.

© Copyright IBM Corp. 2004, 2009

- A memory pointer.

Type definition

```
typedef union Ddpi_Elf_source_s {
  int       source_fd;
  FILE*     source_fp;
  char*     source_mp;
} Ddpi_Elf_Source;
```

Members

source_fd
This integer data type holds the descriptor of the POSIX file, from which the ELF object was loaded.

source_fp
This field holds the pointer to the ANSI C file, from which the ELF object was loaded.

source_mp
This character data type holds the pointer to the memory address from which the ELF object was loaded.

Ddpi_Elf_Source_Type object

The Ddpi_Elf_Source_Type object determines which member of the Ddpi_Elf_Source object is used to track the source of the ELF object.

If a Ddpi_Elf_Source value is not already set, set the Ddpi_Elf_Source_Type to Ddpi_Elf_ST_Unknown.

Type definition

```
typedef enum Ddpi_Elf_Source_Type_s {
   Ddpi_Elf_ST_Unknown = 0,
   Ddpi_Elf_ST_fd      = 1,
   Ddpi_Elf_ST_fp      = 2,
   Ddpi_Elf_ST_mp      = 3
} Ddpi_Elf_Source_Type;
```

Members

Ddpi_Elf_ST_Unknown
If this value is 0, the corresponding Ddpi_Elf_Source does not contain a valid value.

Ddpi_Elf_ST_fd
If this value is 1, the corresponding Ddpi_Elf_Source contains a valid POSIX file descriptor.

Ddpi_Elf_ST_fp
If this value is 2, the corresponding Ddpi_Elf_Source contains a valid ANSI C file pointer.

Ddpi_Elf_ST_mp
If this value is 3, the corresponding Ddpi_Elf_Source contains a valid pointer to a memory block.

ddpi_elf_create operation

The ddpi_elf_create operation creates a Ddpi_Elf object to enable access to debugging data in an ELF object file.

Note: The Ddpi_Elf objects can also be created by the ddpi_module_extract_C_CPP_information operation. This is the preferred operation if the loaded application-executable module was compiled with the z/OS XL C/C++ compiler. The operation creates and lists the Ddpi_Elf objects associated with the given Ddpi_Module object.

Prototype

```
int ddpi_elf_create(
    Ddpi_Access         access,
    Elf*                elf,
    char*               elf_filename,
    Ddpi_Elf_Source     source,
    Ddpi_Elf_Source_Type
                        source_type,
    Dwarf_Addr          ppa2_addr,
    Dwarf_Addr*         func_addr_list,
    Dwarf_Addr*         ppa1_addr_list,
    int                 ppa1_elements,
    Dwarf_Addr          csect_low_addr,
    Dwarf_Addr          csect_high_addr,
    unsigned char       md5_sig[16],
    int                 user_area_len,
    Ddpi_Elf*           ret_elf,
    Ddpi_Error*         error);
```

Parameters

access
Input. This provides access to libddpi.

elf Input. This provides the ELF object file descriptor.

elf_filename
Input. This accepts the ELF object file name.

source
Input. This enables the source pointer to be given to elf_begin.

source_type
Input. This accepts the source_type.

ppa2_addr
Input. This accepts the PPA2 address.

func_addr_list
Input. This accepts a list of function address in the same order as the ppa1_addr_list. The value is copied, and the user may free their version.

ppa1_addr_list
Input. This accepts a list of PPA1 addresses. The value is copied, and the user may free their version.

ppa1_elements
Input. This accepts the number of elements in PPA1 list.

csect_low_addr
Input. This accepts the lowest address in the CSECT (the first byte of the object). Specify `(Dwarf_Addr)-1` for an unknown address. All other addresses will be taken as valid.

csect_high_addr
Input. This accepts the highest address in the CSECT (the last byte of the object). Specify `(Dwarf_Addr)-1` for an unknown address. All other addresses will be taken as valid.

Note: `csect_low_addr` and `csect_high_addr` are not guaranteed to enclose all control blocks. They are guaranteed to enclose all executable code for the given compilation unit (CU) if they are set to a value that is not -1.

md5_sig[16]
Input. This accepts and copies the MD5 signature.

user_area_len
Input. This accepts the user area length.

ret_elf
Output. This returns the `Ddpi_Elf` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the ELF object.

DW_DLV_NO_ENTRY
Returned if the user-area length is zero.

DW_DLV_ERROR
This value is returned if:

- `user_area_len` is less than zero
- An error occurs during memory allocation.

ddpi_elf_term operation

The `ddpi_elf_term` operation terminates a `Ddpi_Elf` object and removes the given `Ddpi_Elf` from the list in its parent access object.

Prototype

```
int ddpi_elf_term(
    Ddpi_Elf           elf,
    Ddpi_Error*        error);
```

Parameters

elf Input. This accepts a `Ddpi_Elf` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful termination of the ELF object.

DW_DLV_NO_ENTRY
Returned if the ELF descriptor within the `Ddpi_Elf` object is not found in the list of the parent `Ddpi_Access` object.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL
- The ELF descriptor within the `Ddpi_Elf` object is NULL
- The `Ddpi_Access` or `Ddpi_Info` object associated with `elf` is NULL
- An error occurs during memory deallocation

ddpi_elf_get_owner operation

The `ddpi_elf_get_owner` operation returns the owner of the `Ddpi_Elf` object.

Prototype

```
int ddpi_elf_get_owner(
      Ddpi_Elf            elf,
      Ddpi_Access*        ret_owner,
      Ddpi_Error*         error);
```

Parameters

access
Input. This accepts a `Ddpi_Elf` object.

ret_owner
Output. This returns the `Ddpi_Access` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the ELF object owner.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `access` is NULL.
- The ELF descriptor within the Ddpi_Elf object is NULL.
- The Ddpi_Info object associated with `access` is NULL.
- `ret_owner` is NULL
- An error occurs during memory deallocation

ddpi_elf_get_source operation

The `ddpi_elf_get_source` operation finds the source data of the Elf object.

Prototype

```
int ddpi_elf_get_source(
  Ddpi_Elf              d_elf,
  Ddpi_Elf_Source*      ret_source,
  Ddpi_Elf_Source_Type*
                        ret_source_type,
  Ddpi_Error*           error);
```

Parameters

d_elf
Input. This accepts a `Ddpi_Elf` object.

ret_source
Output. This returns the source data. This cannot be a NULL value.

ret_source_type
Output. This returns the type of the source data. This cannot be a NULL value.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of source data.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `access` is NULL.
- The ELF descriptor within the `Ddpi_Elf` object is NULL.
- The `Ddpi_Info` or `Ddpi_Info` object associated with `d_elf` is NULL.

ddpi_elf_set_source operation

The `ddpi_elf_set_source` operation stores the value returned by the `elf_begin` operation. The value can be a file descriptor, file pointer, or memory block.

Prototype

```
int ddpi_elf_set_source(
  Ddpi_Elf              d_elf,
  Ddpi_Elf_Source       source,
  Ddpi_Elf_Source_Type
                        source_type,
  Ddpi_Error*           error);
```

Parameters

d_elf
Input. This accepts a `Ddpi_Elf` object.

source
Input. This accepts the source data.

source_type
Input. This accepts the type of source data.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of source data.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `d_elf` is NULL
- The `Ddpi_Info` or `Ddpi_Info` object associated with `d_elf` is NULL.

ddpi_elf_get_elf operation

The `ddpi_elf_get_elf` operation finds the ELF object for the given `Ddpi_Elf` object.

Prototype

```
int ddpi_elf_get_elf(
    Ddpi_Elf            elf,
    Elf**               ret_elf,
    Ddpi_Error*         error);
```

Parameters

elf Input. This accepts a `Ddpi_Elf` object.

ret_elf
Output. This returns the ELF object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the ELF error object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if an error occurs in the parameters.

ddpi_elf_set_elf operation

The `ddpi_elf_set_elf` operation assigns the ELF object to the given `Ddpi_Elf` object.

Prototype

```
int ddpi_elf_set_elf(
    Ddpi_Elf            d_elf,
    Elf*                elf,
    Ddpi_Error*         error);
```

Parameters

d_elf
Input. This accepts a Ddpi_Elf object.

elf Input. This accepts the ELF object that will be set in the Ddpi_Elf object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the error object to the given ELF object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if an error occurs in the parameters.

ddpi_elf_get_elf_file_name operation

The `ddpi_elf_get_elf_file_name` operation returns the pointer to the ELF object filename, and not a copy.

Do not deallocate the pointer.

Prototype

```
int   ddpi_elf_get_elf_file_name(
    Ddpi_Elf            elf,
    char**              ret_elf_filename,
    Ddpi_Error*         error);
```

Parameters

elf Input. This accepts a `Ddpi_Elf` object.

ret_elf_filename
Output. This returns a debug instance. This value cannot be NULL.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the ELF name pointer.

DW_DLV_NO_ENTRY
Returned if there is no stored filename.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL
- The `Ddpi_Access` or `Ddpi_Info` object associated with `elf` is NULL.
- `ret_elf_filename` is NULL.

ddpi_elf_set_elf_file_name operation

The `ddpi_elf_set_elf_file_name` operation assigns an ELF file name to the given `Ddpi_Elf` object and makes a copy of the given name.

Prototype

```
int   ddpi_elf_set_elf_file_name(
    Ddpi_Elf            elf,
    char*               elf_filename,
    Ddpi_Error*         error);
```

Parameters

elf Input. This accepts a `Ddpi_Elf` object.

elf_filename
Input. This accepts a new filename. No validation is done on this value. A NULL filename will cause any previous entry to be replaced with NULL.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the object name.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL
- The `Ddpi_Access` or `Ddpi_Info` object associated with `elf` is NULL.
- `ret_elf_filename` is NULL.
- The function was unable to make a copy of the given filename

ddpi_elf_get_ppa_addrs operation

The `ddpi_elf_get_ppa_addrs` operation queries the PPA addresses for a given `Ddpi_Elf` object.

Only C/C++ compilation units produce PPA control blocks. If the Ddpi_Elf object contains a C/C++ compilation unit, the operation will return one PPA2 address. If the compilation unit is compiled with ISD debug information (in other words, compiled with the TEST compiler option), then the operation will also return the PPA1 and the corresponding function addresses.

Prototype

```
int   ddpi_elf_get_ppa_addrs(
    Ddpi_Elf            elf,
    Dwarf_Addr*         ret_ppa2_addr,
    Dwarf_Addr**        ret_ppa1_addrs,
    Dwarf_Addr**        ret_func_addrs,
    int*                ret_ppa1_count,
    Ddpi_Error*         error);
```

Parameters

elf Input. This accepts a `Ddpi_Elf` object.

ret_ppa2_addr
Output. This returns a PPA2 address of the compilation unit. 0 means that the PPA2 block does not exist.

ret_ppa1_addrs
Output. This returns a pointer to a PPA1 address list. This value cannot be NULL.

ret_func_addrs
Output. This returns a pointer to a function-address list. This value cannot be NULL.

ret_ppa1_count
Output. This returns the number of PPA1 addresses in the list, which is the same number as in the function-address list. This value cannot be NULL. If the returned value is 0, the PPA1 blocks do not exist or the compilation unit is not compiled with the TEST compiler option.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the PPA1 address list.

Note: This does not indicate the presence of PPA2/PPA1 blocks. You must check the individual return value to test for the presence of PPA control blocks.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL or corrupted.
- The `Ddpi_Access` or `Ddpi_Info` object associated with `elf` is NULL.
- `ret_elf_filename` is NULL or corrupted.
- A returned pointer is NULL

ddpi_elf_set_ppa_addrs operation

The `ddpi_elf_set_ppa_addrs` operation assigns the PPA addresses to a given ELF object.

The `ddpi_elf_set_ppa_addrs` operation assigns all four fields each time and overwrites previous values. The values are copied into the `Ddpi_Storage` object. You can deallocate copies of the list.

Prototype

```
int   ddpi_elf_set_ppa_addrs(
    Ddpi_Elf            elf,
    Dwarf_Addr          ppa2_addr,
    Dwarf_Addr*         ppa1_addrs,
    Dwarf_Addr*         func_addrs,
    int                 ppa1_count,
    Ddpi_Error*         error);
```

Parameters

elf Input. This accepts a Ddpi_Elf object.

ppa2_addr
Input. This accepts a PPA2 address.

ppa1_addrs
Input. This accepts a PPA1 address list. There must be `ppa1_count` members in this list.

func_addrs
Input. This accepts a function address list. There must be `ppa1_count` members in this list.

ppa1_count
Input. This accepts the number of PPA1 addresses.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the PPA addresses.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL or corrupted.
- The `Ddpi_Access` or `Ddpi_Info` object associated with `elf` is NULL.

ddpi_elf_get_md5_sig operation

The `ddpi_elf_get_md5_sig` operation copies the MD5 signature into the given unsigned char[16] buffer.

Prototype

```
int   ddpi_elf_get_md5_sig(
    Ddpi_Elf            elf,
    unsigned char       ret_md5_sig[16],
    Ddpi_Error*         error);
```

Parameters

elf Input. This accepts a `Ddpi_Elf` object.

ret_md5_sig[16]
Output. This returns the MD5 signature.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the MD5 address.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL.
- The `Ddpi_Access` or `Ddpi_Info` object associated with `elf` is NULL.

ddpi_elf_set_md5_sig operation

The ddpi_elf_set_md5_sig operation assigns the MD5 signature to a given ELF object. It copies the given unsigned char[16] buffer into the MD5 signature.

Prototype

```
int   ddpi_elf_set_md5_sig(
    Ddpi_Elf            elf,
    unsigned char       md5_sig[16],
    Ddpi_Error*         error);
```

Parameters

elf Input. This accepts a Ddpi_Elf object.

md5_sig[16]
Input. This accepts a MD5 signature.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the MD5 address.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- elf is NULL.
- The Ddpi_Access or Ddpi_Info object associated with elf is NULL.

ddpi_elf_get_csect_addrs operation

The ddpi_elf_get_csect_addrs operation queries the CSECT addresses for the ELF object and extracts the high and low addresses.

ddpi_elf_get_csect_addrs function helps to select and load a CU when there are multiple CUs from which to choose.

Prototype

```
int ddpi_elf_get_csect_addrs(
  Ddpi_Elf              d_elf,
  Dwarf_Addr*           ret_low_addr,
  Dwarf_Addr*           ret_high_addr,
  Ddpi_Error*           error);
```

Parameters

d_elf
Input. This accepts a Ddpi_Elf object.

ret_low_addr
Output. This returns the lowest known address in CSECT (the first byte of the object). (Dwarf_Addr)-1 is not an acceptable value address.

ret_high_addr
Output. This returns the highest known address in CSECT (the last byte of the object). (Dwarf_Addr)-1 is not an acceptable value address.

Note: ret_low_addr and ret_high_addr are not guaranteed to enclose all control blocks. They are guaranteed to enclose all executable code for the given CU if they are set to a value that is not -1.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the high and low addresses.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- elf is NULL.
- The Ddpi_Access or Ddpi_Info object associated with elf is NULL.

ddpi_elf_set_csect_addrs operation

The ddpi_elf_set_csect_addrs operation assigns the high and low addresses to the CSECT of the ELF object.

ddpi_elf_set_csect_addrs should be used for contiguous CUs only. These values help to determine the owning CU of a given address. That is, it is used to determine which CU to load, if there are multiple CUs.

An address of (Dwarf_Addr)-1 is considered a bad address. csect_low_addr points to the beginning of the object. csect_high_addr points to the last byte in the object. csect_low_addr and csect_high_addr are not guaranteed to enclose all control blocks. If they are set to a non -1 value, they are guaranteed to enclose all executable code for the given CU.

Prototype

```
int ddpi_elf_set_csect_addrs(
  Ddpi_Elf              d_elf,
  Dwarf_Addr            low_addr,
  Dwarf_Addr            high_addr,
  Ddpi_Error*           error);
```

Parameters

d_elf
Input. This accepts a Ddpi_Elf object.

low_addr
Input. This accepts the lowest known address in CSECT (the first byte of the object). (Dwarf_Addr)-1 is not an acceptable value address.

high_addr
Input. This accepts the highest known address in CSECT (the last byte of the object). (Dwarf_Addr)-1 is not an acceptable value address.

Note: low_addr and high_addr are not guaranteed to enclose all control blocks. They are guaranteed to enclose all executable code for the given CU if they are set to a value that is not -1.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the high and low addresses.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL.
- The `Ddpi_Access` or `Ddpi_Info` object associated with `elf` is NULL.

ddpi_elf_get_user_area operation

The `ddpi_elf_get_user_area` operation returns a pointer to the user area for the `Ddpi_Elf` object.

Do not deallocate the pointer. It is not a copy.

Prototype

```
int   ddpi_elf_get_user_area(
    Ddpi_Elf            elf,
    Dwarf_Ptr*          ret_user_area,
    Ddpi_Error*         error);
```

Parameters

elf Input. This accepts a `Ddpi_Elf` object.

ret_user_area
Output. This returns the user area of the `Ddpi_Elf` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the pointer to the user area.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL.
- The Ddpi_Access or Ddpi_Info object associated with `elf` is NULL.

ddpi_elf_list_function operation

The `ddpi_elf_list_function` operation lists all of the `Ddpi_Function` objects associated with the given `Ddpi_Elf` object. Although the calling module must deallocate the list, the individual `Ddpi_Function` objects in the list should never be deallocated, as these are the actual `Ddpi_Function` objects from the `Ddpi_Elf` object.

The calling function should deallocate the returned list by using the following operation:

```
ddpi_dealloc(info, *ret_functions, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_elf_list_function(
  Ddpi_Elf              elf,
  Ddpi_Function**       ret_functions,
  Dwarf_Unsigned*       ret_count,
  Ddpi_Error*           error);
```

Parameters

elf Input. This accepts the `Ddpi_Elf` object.

ret_functions
Output. This returns the list of `Ddpi_Function` objects in the `Ddpi_Elf` object as an array.

ret_count
Output. This returns the number of `Ddpi_Function` objects in the array.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
There are no Ddpi_Function objects associated with the given Ddpi_Elf object.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL or invalid.
- The Ddpi_Info, Ddpi_Access, or Ddpi_Module object associated with the given Ddpi_Elf object is NULL or invalid
- The `ret_functions` array is NULL.
- The `ret_count` value is NULL.
- An error occurs during memory allocation.

ddpi_elf_list_variable operation

The `ddpi_elf_list_variable` operation lists all of the `Ddpi_Variable` objects associated with the given `Ddpi_Elf` object. Although the calling module must deallocate the list, the individual `Ddpi_Variable` objects in the list should never be deallocated, as these are the actual `Ddpi_Variable` objects associated with the `Ddpi_Elf` object.

The calling function should deallocate the returned list by using the following operation:

```
ddpi_dealloc(info, *ret_variables, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_elf_list_variable(
   Ddpi_Elf             elf,
   Ddpi_Variable**      ret_variables,
   Dwarf_Unsigned*      ret_count,
   Ddpi_Error*          error);
```

Parameters

elf Input. This accepts the given `Ddpi_Elf` object.

ret_functions
Output. This returns the list of `Ddpi_Variable` objects in the `Ddpi_Elf` object as an array.

ret_count
Output. This returns the number of `Ddpi_Variable` objects in the array.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
There are no Ddpi_Function objects associated with the given `Ddpi_Elf` object.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL or invalid.
- The `Ddpi_Info`, `Ddpi_Access`, or `Ddpi_Module` object associated with the given `Ddpi_Elf` object is NULL or invalid
- The `ret_variables` array is NULL.
- The `ret_count` value is NULL.
- An error occurs during memory allocation.

ddpi_elf_list_type operation

The `ddpi_elf_list_type` operation creates an array that contains all of the `Ddpi_Type` objects associated with the given `Ddpi_Elf` object. Although the calling module must deallocate the list, the individual `Ddpi_Variable` objects in the list should never be deallocated; these are the actual `Ddpi_Type` objects associated with the `Ddpi_Elf` object.

The calling function should deallocate the returned array by using the following operation:

```
ddpi_dealloc(info, *ret_types, DDPI_DLA_LIST);
```

Prototype

```
 int ddpi_elf_list_type(
   Ddpi_Elf             elf,
   Ddpi_Type**          ret_types,
   Dwarf_Unsigned*      ret_count,
   Ddpi_Error*          error);
```

Parameters

elf Input. This accepts the given Ddpi_Elf object.

ret_types
Output. This returns the list of Ddpi_Type objects in the Ddpi_Elf object as an array.

ret_count
Output. This returns the number of Ddpi_Type objects in the array.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
There are no Ddpi_Type objects associated with the given Ddpi_Elf object.

DW_DLV_ERROR
This value is returned if:

- elf is NULL or invalid.
- The Ddpi_Info, Ddpi_Access, or Ddpi_Module object associated with the given Ddpi_Elf object is NULL or invalid
- The ret_types array is NULL.
- The ret_count value is NULL.
- An error occurs during memory allocation.

ddpi_elf_list_sourcefile operation

The ddpi_elf_list_sourcefile operation creates an array that contains all of the Ddpi_Sourcefile objects associated with the given Ddpi_Elf object. Although the calling module must deallocate the list, the individual Ddpi_Sourcefile objects in the list should never be deallocated; these are the actual Ddpi_Sourcefile objects associated with the Ddpi_Elf object.

The calling function should deallocate the returned array by using the following operation:

```
ddpi_dealloc(info, *ret_sourcefiles, DDPI_DLA_LIST);
```

Prototype

```
int ddpi_elf_list_sourcefile(
   Ddpi_Elf              elf,
   Ddpi_Sourcefile**     ret_sourcefiles,
   Dwarf_Unsigned*       ret_count,
   Ddpi_Error*           error);
```

Parameters

elf Input. This accepts the given Ddpi_Elf object.

ret_sourcefiles
Output. This returns the list of Ddpi_Sourcefile objects in the Ddpi_Elf object as an array.

ret_count
Output. This returns the number of Ddpi_Sourcefile objects in the array.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is returned if there are no Ddpi_Sourcefile objects associated with the given Ddpi_Elf object.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL or invalid.
- The Ddpi_Info, Ddpi_Access, or Ddpi_Module object associated with the given Ddpi_Elf object is NULL or invalid
- The `ret_types` array is NULL.
- The `ret_count` value is NULL.
- An error occurs during memory allocation.

ddpi_elf_get_primary_sourcefile operation

The `ddpi_elf_get_primary_sourcefile` operation returns the primary `Ddpi_Sourcefile` object associated with the given `Ddpi_Elf` object. The `Ddpi_Sourcefile` object should never be deallocated; this is the actual primary `Ddpi_Sourcefile` object associated with the `Ddpi_Elf` object.

Prototype

```
int ddpi_elf_get_primary_sourcefile(
   Ddpi_Elf              elf,
   Ddpi_Sourcefile*      ret_sourcefile,
   Ddpi_Error*           error);
```

Parameters

elf Input. This accepts the given `Ddpi_Elf` object.

ret_sourcefile
Output. This returns the primary `Ddpi_Sourcefile` object for the `Ddpi_Elf` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
The primary `Ddpi_Sourcefile` object is not available to the given `Ddpi_Elf` object.

DW_DLV_ERROR
This value is returned if:

- `elf` is NULL or invalid.
- The `Ddpi_Info`, `Ddpi_Access`, or `Ddpi_Module` object associated with the given `Ddpi_Elf` object is NULL or invalid
- The `ret_sourcefile` array is NULL.

- An error occurs during memory allocation.

Chapter 17. Ddpi_Class APIs

A Ddpi_Class object is used to map certain portions of memory to certain uses. For example, it can be used to allocate space for program code, a Writable Static Area (WSA), a heap, or a stack. It can be mapped to a `binder` or `goff` class.

A class can be owned by one of the following three different entities: a module, a process, or a thread. Typically, these entities are owned at the module level (WSA, program_code, B_LIT), but some entities can be owned at the process level (heap), or the thread level (stack).

Ddpi_Class_Type object

The class type.

Type definition

```
typedef enum Ddpi_Class_Type_s {
  Ddpi_CT_Unknown       =  0,
  Ddpi_CT_B_LIT         =  1,
  Ddpi_CT_Program_code  =  2,
  Ddpi_CT_WSA           =  3,
  Ddpi_CT_Heap_Seg      =  4,
  Ddpi_CT_Stack_Seg     =  5,
  Ddpi_CT_Other         =  6
} Ddpi_Class_Type;
```

Members

Ddpi_CT_Unknown
If this value is 0, the mapped usage of the associated memory is unknown.

Ddpi_CT_B_LIT
If this value is 1, the associated memory is mapped for literal values.

Ddpi_CT_Program_code
If this value is 2, the associated memory is mapped for executable code.

Ddpi_CT_WSA
If this value is 3, the associated memory is mapped for writable static area containing program variables.

Ddpi_CT_Heap_Seg
If this value is 4, the associated memory is mapped for heap storage.

Ddpi_CT_Stack_Seg
If this value is 5, the associated memory is mapped for stack storage.

Ddpi_CT_Other
If this value is 6, the associated memory is mapped for storage of data that is not specified in the type definition.

ddpi_class_get_storage_attr operation

The `ddpi_class_get_storage_attr` operation returns the storage attribute of a given `Ddpi_Class` object.

© Copyright IBM Corp. 2004, 2009

Prototype

```
int ddpi_class_get_storage_attr(
  Ddpi_Class             class_obj, /
  Ddpi_Class_Storage_Attr*
                         ret_attr,
  Ddpi_Error*            error);
```

Parameters

class_obj
Input. This accepts the `Ddpi_Class` object.

ret_attr
Output. This returns the storage attribute.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the class storage attribute.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `class_obj` is NULL.
- The `Ddpi_Info` object associated with `class_obj` is NULL.
- `ret_attr` is NULL.

Ddpi_Class_Owner_Type object

This is used to describe which libddpi entity is the parent/owner of the associated Ddpi_Class_Owner object.

Type definition

```
typedef enum Ddpi_Class_Owner_Type_s {
  Ddpi_Class_Owner_Unknown =  0,
  Ddpi_Class_Owner_Thread  =  1,
  Ddpi_Class_Owner_Module  =  2,
  Ddpi_Class_Owner_Process =  3
} Ddpi_Class_Owner_Type;
```

Members

Ddpi_Class_Owner_Unknown
0

Ddpi_Class_Owner_Thread
1

Ddpi_Class_Owner_Module
2

Ddpi_Class_Owner_Process
3

Ddpi_Class_Owner object

This object is used to hold the owner of the given Ddpi_Class object. The associated Ddpi_Class_Owner_Type object is used to determine which union member should be accessed.

Type definition

```
typedef union Ddpi_Class_Owner_s {
  Ddpi_Thread     owner_thread;
  Ddpi_Module     owner_module;
  Ddpi_Process    owner_process;
} Ddpi_Class_Owner;
```

Members

owner_thread
Ddpi_Thread

owner_module
Ddpi_Module

owner_process
Ddpi_Process

Ddpi_Class object

The Ddpi_Class object is an opaque data type that contains information regarding a single class of storage (using the binder and GOFF terminology) for a given application-executable module.

Type definition

```
typedef struct Ddpi_Class_s*      Ddpi_Class;
```

ddpi_class_create operation

The `ddpi_class_create` operation creates a `Ddpi_Class` object to describe a typed set of storage ranges and returns a pointer to the object.

When the user calls this function, passing a character string as the parameter name, the operation copies the content of name and stores the copy. After this function returns, you can deallocate the original name and save storage.

Prototype

```
int ddpi_class_create(
  Ddpi_Class_Owner      owner,
  Ddpi_Class_Owner_Type owner_type,
  Ddpi_Class_Type       type,
  char*                 name,
  Dwarf_Addr            addr_low,
  Dwarf_Addr            addr_high,
  Ddpi_Class_Storage_Attr
                        attr,
  int                   user_area_len,
  Ddpi_Class*           ret_class,
  Ddpi_Error*           error);
```

Parameters

owner
Input. This accepts the class owner.

owner_type
Input. This accepts the class owner type.

type
Input. This accepts the class type.

name
Input. This accepts the class name string.

addr_low
Input. This accepts the class start address (the first byte in the class).

addr_high
Input. This accepts the class end address (the last byte in the class).

attr
Input. This accepts the class storage attribute.

user_area_len
Input. This accepts the user area length.

ret_class
Output. This returns the `Ddpi_Class` pointer.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the class pointer.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `owner_type` is not set to one of Ddpi_Class_Owner_Module, `Ddpi_Class_Owner_Thread`, or `Ddpi_Class_Owner_Process` objects.
- `owner` is NULL.
- The `Ddpi_Space` or `Ddpi_Info` object associated with `owner` is NULL.
- `ret_class` is NULL.
- An error occurs during memory allocation.
- `user_area_len` is less than zero.
- `ret_class` overlaps an existing class.

  Note: Only one class for a given address is allowed in a given address space. A class may overlap if it has exactly the same values for `type`, `name`, `addr_low`, `addr_high`, and `attr`.

ddpi_class_term operation

The `ddpi_class_term` operation releases all internal resources associated with the descriptor class object, and invalidates the object.

The `ddpi_class_term` operation terminates the `Ddpi_Section` children of the given class type.

Note: Termination of subcomponents is not done until all instances of the same address range/class have been terminated.

Prototype

```
int ddpi_class_term(
  Ddpi_Class            class_obj,
  Ddpi_Error*           error);
```

Parameters

class_obj
Input. This accepts the `Ddpi_Class` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of all internal resources associated with the descriptor class object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `class_obj` is NULL or not of a valid type
- The `Ddpi_Info` object associated with `class_obj` is NULL
- An error occurs during termination of child descriptors
- An error occurs during memory allocation.
- `ret_class` is NULL

ddpi_class_get_owner operation

The `ddpi_class_get_owner` operation returns the owner and owner type of a given class object.

Prototype

```
int ddpi_class_get_owner(
  Ddpi_Class            class_obj,
  Ddpi_Class_Owner*     ret_owner,
  Ddpi_Class_Owner_Type*
                        ret_owner_type,
  Ddpi_Error*           error);
```

Parameters

class_obj
Input. This accepts the `Ddpi_Class` object.

ret_owner
Output. This returns the class owner.

ret_owner_type
Output. This returns the class owner type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the class owner and owner type.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `class_obj` is NULL.
- The `Ddpi_Info` object associated with `class_obj` is NULL.

ddpi_class_get_name operation

The `ddpi_class_get_name` operation finds the name of a given `Ddpi_Class` object and sets the returned name to a pointer to a null-terminated string of characters.

`ddpi_class_get_name` returns the actual version of the name, not a copy. Never deallocate the returned pointer.

Prototype

```
int ddpi_class_get_name(
  Ddpi_Class            class_obj,
  char**                ret_name,
  Ddpi_Error*           error);
```

Parameters

class_obj
Input. This accepts the `Ddpi_Class` object.

ret_name
Output. This returns the `name` string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the name pointer.

DW_DLV_NO_ENTRY
Returned if the `name` string is NULL.

DW_DLV_ERROR
This value is returned if:

- `class_obj` is NULL.
- The `Ddpi_Info` object associated with `class_obj` is NULL.

ddpi_class_set_name operation

The `ddpi_class_set_name` operation assigns a new name to a given `Ddpi_Class` object.

`ddpi_class_set_name` can be used to set the name to NULL. It copies the given name. The caller may deallocate `new_name` after the call to save memory.

Prototype

```
int ddpi_class_set_name(
  Ddpi_Class            class_obj,
  char*                 new_name,
  Ddpi_Error*           error);
```

Parameters

class_obj
Input. This accepts the `Ddpi_Class` object.

new_name
Input. This accepts the new class name.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the new class name.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `class_obj` is NULL
- The `Ddpi_Info` object associated with `class_obj` is NULL.
- An error occurs during memory allocation for the copy of the `name` string.

ddpi_class_get_type operation

The `ddpi_class_get_type` operation returns the type of the given `Ddpi_Class` object.

Prototype

```
int ddpi_class_get_type(
  Ddpi_Class            class_obj,
  Ddpi_Class_Type*      ret_type,
  Ddpi_Error*           error);
```

Parameters

class_obj
Input. This accepts the Ddpi_Class object.

ret_type
Output. This returns the class type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the class type.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- class_obj is NULL
- The Ddpi_Info object associated with class_obj is NULL.
- ret_type is NULL

ddpi_class_get_storage_attr operation

The ddpi_class_get_storage_attr operation returns the storage attribute of a given Ddpi_Class object.

Prototype

```
int ddpi_class_get_storage_attr(
  Ddpi_Class             class_obj, /
  Ddpi_Class_Storage_Attr*
                         ret_attr,
  Ddpi_Error*            error);
```

Parameters

class_obj
Input. This accepts the Ddpi_Class object.

ret_attr
Output. This returns the storage attribute.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the class storage attribute.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- class_obj is NULL.
- The Ddpi_Info object associated with class_obj is NULL.
- ret_attr is NULL.

ddpi_class_get_addr_low operation

The ddpi_class_get_addr_low operation returns the start address of a given class object.

Prototype

```
int ddpi_class_get_addr_low(
  Ddpi_Class            class_obj,
  Dwarf_Addr*           ret_addr_low,
  Ddpi_Error*           error);
```

Parameters

class_obj
Input. This accepts the Ddpi_Class object.

ret_addr_low
Output. This returns the starting address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the class storage address.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `class_obj` is NULL.
- The `Ddpi_Info` object associated with `class_obj` is NULL.
- `ret_addr_low` is NULL

ddpi_class_get_addr_high operation

The `ddpi_class_get_addr_high` operation returns the end address of a given class object.

Prototype

```
int ddpi_class_get_addr_high(
  Ddpi_Class            class_obj,
  Dwarf_Addr*           ret_addr_high,
  Ddpi_Error*           error);
```

Parameters

class_obj
Input. This accepts the `address`.

ret_addr_high
Output. This returns the end address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of resources associated with the `Ddpi_Info` descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `class_obj` is NULL.
- The `Ddpi_Info` object associated with `class_obj` is NULL.
- `ret_addr_high` is NULL

ddpi_class_get_user_area operation

The `ddpi_class_get_user_area` operation returns the user area of a given class object.

Prototype

```
int ddpi_class_get_user_area(
  Ddpi_Class            class_obj,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

class_obj
Input. This accepts the `Ddpi_Class` object.

ret_user_area
Output. This returns the user area.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of resources associated with the `Ddpi_Info` descriptor.

DW_DLV_NO_ENTRY
Returned if the user-area length is zero.

DW_DLV_ERROR
This value is returned if:

- `class_obj` is NULL.
- The `Ddpi_Info` object associated with `class_obj` is NULL.

ddpi_class_list_section operation

The `ddpi_class_list_section` operation lists all `Ddpi_Section` objects associated with the given `Ddpi_Class` object.

It sets the list to point to `Ddpi_Section` descriptors and the number of Ddpi_Section objects.

When user processing is complete, you must call `ddpi_dealloc` with type DDPI_DLA_LIST to deallocate the list storage.

Prototype

```
int ddpi_class_list_section(
  Ddpi_Class            class_obj,
  Ddpi_Section**        ret_section_list,
  Dwarf_Signed*         ret_section_cnt,
  Ddpi_Error*           error);
```

Parameters

class_obj
Input. This accepts the `Ddpi_Class` object.

ret_section_list
Output. This returns the section list.

ret_section_cnt
Output. This returns and sets the count of the entries in the section list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the class section list.

DW_DLV_NO_ENTRY
Returned if the retrieved section list is empty.

DW_DLV_ERROR
This value is returned if:

- `class_obj` is NULL.
- The `Ddpi_Info` object associated with `class_obj` is NULL.
- `ret_section_list` or `ret_section_cnt` is NULL.
- An error occurs during memory allocation for the list.

Chapter 18. Ddpi_Section APIs

You can use Ddpi_Section APIs to divide a class to smaller sections, such as for a pseudoregister (PR) or a control section (CSECT).

Ddpi_Section opaque object

The Ddpi_Section object is an opaque data type that contains information about application-executable modules.

Type definition

```
typedef struct Ddpi_Section_s*    Ddpi_Section;
```

ddpi_section_create operation

The `ddpi_section_create` operation creates a `Ddpi_Section` object to describe a range of storage.

When the user calls the `ddpi_section_create` operation, passing a character string as the parameter name, the operation copies the content of name and sets `section->ds_name` to the copy. You can deallocate the copy.

If the parent class has the same attributes as another class (such as name, type, and location), this section is applied to both instances of the class.

Prototype

```
int ddpi_section_create(
  Ddpi_Class            class_obj,
  char *                name,
  Dwarf_Addr            addr_low,
  Dwarf_Addr            addr_high,
  int                   user_area_len,
  Ddpi_Section*         ret_section,
  Ddpi_Error*           error);
```

Parameters

class_obj
Input. This accepts the `Ddpi_Class` object.

name
Input. This accepts the section name.

addr_low
Input. This accepts the section start address

addr_high
Input. This accepts the section end address.

user_area_len
Input. This accepts the user area length.

ret_section
Output. This returns the `Ddpi_Section` object.

error
See “The libddpi error parameter” on page 13.

© Copyright IBM Corp. 2004, 2009

Return values

DW_DLV_OK
Returned upon successful creation of the section copy.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `class_obj` is NULL
- The `Ddpi_Info` object associated with `class_obj` is NULL.
- `ret_section` is NULL.
- An error occurs during allocation of memory for the new section.
- An error occurs during allocation of memory for the copy of the name.
- An error occurs during allocation of memory for the section list of the parent class.
- `user_area_len` is less than zero.

ddpi_section_term operation

The `ddpi_section_term` operation releases all internal resources associated with the descriptor `section`, and invalidates `section`.

Prototype

```
int ddpi_section_term(
  Ddpi_Section          section,
  Ddpi_Error*           error);
```

Parameters

section
Input. This accepts the `Ddpi_Section` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of resources associated with the section descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `section` is NULL
- The `Ddpi_Info` object associated with `section` is NULL.
- An error occurs during allocation of memory.

The `ddpi_section_term` operation returns `DW_DLV_OK` The `ddpi_section_term` operation returns `DW_DLV_ERROR` if:

ddpi_section_get_owner operation

The ddpi_section_get_owner operation queries the owner of a given Ddpi_Section object.

Prototype

```
int ddpi_section_get_owner(
  Ddpi_Section          section,
  Ddpi_Class*           ret_owner,
  Ddpi_Error*           error);
```

Parameters

section
Input. This accepts the Ddpi_Section object.

ret_owner
Output. This returns the section owner.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of resources associated with the Ddpi_Info descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- section or its corresponding Ddpi_Info object is NULL.
- ret_owner is NULL.

ddpi_section_get_name operation

The ddpi_section_get_name operation finds the name of a given Ddpi_Section object and sets the returned name to a pointer to a null-terminated string of characters.

The name string is part of the Ddpi_Section object, and must not be deallocated by the caller.

Prototype

```
int ddpi_section_get_name(
  Ddpi_Section          section,
  char**                ret_name,
  Ddpi_Error*           error);
```

Parameters

section
Input. This accepts the Ddpi_Section object.

ret_name
Output. This returns the section name.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the section name pointer.

DW_DLV_NO_ENTRY
Returned if the retrieved name is NULL.

DW_DLV_ERROR
This value is returned if:

- `section` or its corresponding `Ddpi_Info` object is NULL.
- `ret_name` is NULL.

ddpi_section_set_name operation

The `ddpi_section_set_name` operation assigns a new name to a given `Ddpi_Section` object.

Prototype

```
int ddpi_section_set_name(
  Ddpi_Section          section,
  char*                 new_name,
  Ddpi_Error*           error);
```

Parameters

section
Input. This accepts the `Ddpi_Section` object.

new_name
Input. This accepts the new section name.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of resources associated with the `Ddpi_Info` descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `section` or its corresponding `Ddpi_Info` object is NULL.
- An error occurs while allocating storage.

ddpi_section_get_addr_low operation

The `ddpi_section_get_addr_low` operation returns the start address of a given `Ddpi_Section` object.

Prototype

```
int ddpi_section_get_addr_low(
  Ddpi_Section          section,
  Dwarf_Addr*           ret_addr_low,
  Ddpi_Error*           error);
```

Parameters

section
Input. This accepts the `Ddpi_Section` object.

ret_addr_low
Output. This returns the start address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the section ID start address.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `section` or its corresponding `Ddpi_Info` object is NULL.
- `ret_addr_low` is NULL.

ddpi_section_get_addr_high operation

The `ddpi_section_get_addr_high` operation queries the end address of a given `Ddpi_Section` object, where `addr_high` is the address of the last byte in the section.

Prototype

```
int ddpi_section_get_addr_high(
  Ddpi_Section          section,
  Dwarf_Addr*           ret_addr_high,
  Ddpi_Error*           error);
```

Parameters

section
Input. This accepts the `Ddpi_Section` object.

ret_addr_high
Output. This returns the start address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the section ID end address.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- section or its corresponding Ddpi_Info object is NULL.
- ret_addr_high is NULL.

ddpi_section_set_addr operation

The ddpi_section_set_addr operation assigns the start and end addresses of a given Ddpi_Section object.

Prototype

```
int ddpi_section_set_addr(
  Ddpi_Section          section,
  Dwarf_Addr            addr_low,
  Dwarf_Addr            addr_high,
  Ddpi_Error*           error);
```

Parameters

section
Input. This accepts the Ddpi_Section object.

addr_low
Input. This accepts the new start address.

addr_high
Input. This accepts the new end address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the section address.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- section or its corresponding Ddpi_Info object is NULL.
- addr_low or addr_high extends outside the address range of the owning class.

ddpi_section_get_user_area operation

The ddpi_section_get_user_area operation returns a pointer to the user area allocated for the given Ddpi_Section object.

Prototype

```
int ddpi_section_get_user_area(
  Ddpi_Section          section,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

section
Input. This accepts the Ddpi_Section object.

ret_user_area
Output. This returns the section user area.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the pointer to the user area.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `section` or its corresponding `Ddpi_Info` object is NULL.
- The address extends outside the address range of the owning class.

Chapter 19. Ddpi_Function APIs

Ddpi_Function APIs find and extract information about a specific function, including static functions. Each Ddpi_Function object is owned by a Ddpi_Elf object. A ddpi_function operation queries one or more Ddpi_Function objects and extracts information about the specific function.

Ddpi_Function object

Each Ddpi_Function object is an opaque data type that contains information about a specific function, including static functions. If the function is defined in more than one compilation unit, there will be a separate Ddpi_Function object for each compilation unit.

A Ddpi_Function object can be queried to get:

- The fully qualified name of the function.
- The unqualified name of the function.
- The Ddpi_Access object that identifies the .dbg file for the function.

If the function is not a member of a C++ class, the fully qualified name will be the same as the unqualified name.

Type definition

```
typedef struct Ddpi_Function_s* Ddpi_Function;
```

ddpi_function_get_full_name operation

The ddpi_function_get_full_name operation returns the fully qualified name (which is prefixed with the C++ class name, if applicable) of the function. The actual version of the name is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_function_get_full_name(
  Ddpi_Function         function,
  char**                ret_full_name,
  Ddpi_Error*           error);
```

Parameters

function
Input. This accepts the Ddpi_Function object for the function.

ret_full_name
Output. This returns the fully qualified name of the function.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

© Copyright IBM Corp. 2004, 2009

DW_DLV_ERROR
This value is returned if:

- `function` is NULL or invalid.
- The Ddpi_Elf, Ddpi_Access or Ddpi_Info object associated with the function is either NULL or invalid.
- `ret_full_name` is NULL.

ddpi_function_get_short_name operation

The `ddpi_function_get_short_name` operation returns the unqualified name which is not prefixed with a C++ class name) of the function. The actual version of the name is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_function_get_short_name(
   Ddpi_Function         function,
   char**                ret_short_name,
   Ddpi_Error*           error);
```

Members

function
Input. This accepts the `Ddpi_Function` object for the function.

ret_short_name
Output. This returns the unqualified name of the function.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `function` is NULL or invalid.
- The `Ddpi_Elf`, `Ddpi_Access`, or `Ddpi_Info` object associated with the function is either NULL or invalid.
- `ret_short_name` is NULL.

ddpi_function_get_access operation

The `ddpi_function_get_access` operation returns the `Ddpi_Access` object whose `Dwarf_Debug` object contains the function instance. The actual `Ddpi_Access` object is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_function_get_access(
   Ddpi_Function         function,
   Ddpi_Access*          ret_access,
   Ddpi_Error*           error);
```

Parameters

function
Input. This accepts the `Ddpi_Function` object for the function.

ret_access
Output. This returns the `Ddpi_Access` object whose `Dwarf_Debug` object contains the function instance.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `function` is NULL or invalid.
- The Ddpi_Elf, Ddpi_Access or Ddpi_Info object associated with the function is either NULL or invalid.
- `ret_full_name` is NULL.

ddpi_function_get_elf operation

The `ddpi_function_get_elf` operation returns the `Ddpi_Elf` object that owns the function instance. The actual `Ddpi_Elf` object is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_function_get_elf(
  Ddpi_Function           function,
  Ddpi_Elf*            ret_elf,
  Ddpi_Error*          error);
```

Members

function
Input. This accepts the `Ddpi_Function` object for the function.

ret_elf
Output. This returns the `Ddpi_Elf` object that owns the function.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `function` is NULL or invalid.

- `ret_elf` is NULL.

ddpi_function_get_die_offset operation

The `ddpi_function_get_die_offset` operation returns the DIE offset of the function.

Prototype

```
int ddpi_function_get_die_offset(
   Ddpi_Function         function,
   Dwarf_Off*            ret_offset,
   Ddpi_Error*           error);
```

Parameters

function
Input. This accepts the `Ddpi_Function` object for the function.

ret_offset
Output. This returns the DIE offset for the function.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `function` is NULL or invalid.
- `ret_full_name` is NULL.

Chapter 20. Ddpi_Variable APIs

Ddpi_Variable APIs provide information about global variables in the .debug_info section.

Ddpi_Variable object

A Ddpi_Variable object contains information about a global variable, including global static variables and static members. If this variable is in more than one compilation unit, there will be a separate Ddpi_Variable object for each unit.

A Ddpi_Variable object can be queried to get the following:

- The fully qualified name (which is prefixed with the C++ class name if applicable).
- The unqualified name.
- The Ddpi_Access object whose .dbg file contains the variable.

If the variable is not nested within a C++ class, the fully qualified name will be the same as the unqualified name.

Type definition

```
typedef struct Ddpi_Variable_s* Ddpi_Variable;
```

ddpi_variable_get_full_name operation

The ddpi_variable_get_full_name operation returns the fully qualified name (which is prefixed with the C++ class name, if applicable) of the variable. The actual version of the name is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_variable_get_full_name(
   Ddpi_Variable         variable,
   char**                ret_full_name,
   Ddpi_Error*           error);
```

Parameters

variable
Input. This accepts the Ddpi_Variable object for the variable.

ret_full_name
Output. This returns the fully qualified name of the variable.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

© Copyright IBM Corp. 2004, 2009

- `variable` is NULL or invalid.
- The `Ddpi_Elf`, `Ddpi_Access`, or `Ddpi_Info` object associated with the variable is either NULL or invalid.
- `ret_full_name` is NULL.

ddpi_variable_get_short_name operation

The `ddpi_variable_get_short_name` operation returns the unqualified name (which is not prefixed with a C++ class name) of the variable. The actual version of the name is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_variable_get_short_name(
   Ddpi_Variable         variable,
   char**                ret_short_name,
   Ddpi_Error*           error);
```

Parameters

variable
Input. This accepts the `Ddpi_Variable` object for the variable.

ret_short_name
Output. This returns the unqualified name of the variable.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `variable` is NULL or invalid.
- The `Ddpi_Elf`, `Ddpi_Access`, or `Ddpi_Info` object associated with the variable is either NULL or invalid.
- `ret_short_name` is NULL.

ddpi_variable_get_access operation

The `ddpi_variable_get_access` operation returns the `Ddpi_Access` object whose `Dwarf_Debug` object contains the variable instance. The actual `Ddpi_Access` object is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_variable_get_access(
   Ddpi_Variable         variable,
   Ddpi_Access*          ret_access,
   Ddpi_Error*           error);
```

Parameters

variable
Input. This accepts the `Ddpi_Variable` object for the variable.

ret_access
Output. This returns the `Ddpi_Access` object whose `Dwarf_Debug` object contains the variable instance.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `variable` is NULL or invalid.
- The `Ddpi_Elf`, `Ddpi_Access`, or `Ddpi_Info` object associated with the variable is either NULL or invalid.
- `ret_access` is NULL.

ddpi_variable_get_die_offset operation

The `ddpi_variable_get_die_offset` operation returns the DIE offset of the variable in the `.debug_info` section.

Prototype

```
int ddpi_variable_get_die_offset(
   Ddpi_Variable         variable,
   Dwarf_Off*            ret_offset,
   Ddpi_Error*           error);
```

Members

variable
Input. This accepts the `Ddpi_Variable` object for the variable.

ret_offset
Output. This returns the DIE offset for the variable.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `variable` is NULL or invalid.
- `ret_offset` is NULL.

Chapter 21. Ddpi_Type APIs

Ddpi_Type APIs provide information about external types in the .debug_info section.

Ddpi_Type object

A Ddpi_Type object contains information about an external type. If this type is in more than one compilation unit, there will be a separate Ddpi_Type object for each unit.

This object can be queried for the following information:

- The fully qualified name (which is prefixed with the C++ class name if applicable).
- The unqualified name.
- The Ddpi_Access object whose .dbg file contains the external type.

If the type is not nested within a C++ class, the fully qualified name will be the same as the unqualified name.

Type definition

```
typedef struct Ddpi_Type_s* Ddpi_Type;
```

ddpi_type_get_access operation

The ddpi_type_get_access operation returns the Ddpi_Access object whose Dwarf_Debug object contains the type instance. The actual Ddpi_Access object is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_type_get_access(
  Ddpi_Type             type,
  Ddpi_Access*          ret_access,
  Ddpi_Error*           error);
```

Parameters

type
Input. This accepts the Ddpi_Type object for the type.

ret_access
Output. This returns the Ddpi_Access object whose Dwarf_Debug object contains the type instance.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

© Copyright IBM Corp. 2004, 2009

- type is NULL or invalid.
- The Ddpi_Elf, Ddpi_Access or Ddpi_Info object associated with the type is either NULL or invalid.
- ret_full_name is NULL.

ddpi_type_get_elf operation

The ddpi_type_get_elf operation returns the Ddpi_Elf object that owns the Ddpi_Type object for the type instance. The actual Ddpi_Elf object is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_type_get_elf(
  Ddpi_Type             type,
  Ddpi_Elf*             ret_elf,
  Ddpi_Error*           error);
```

Parameters

type
Input. This accepts the Ddpi_Type object for the type instance.

ret_elf
Output. This returns the Ddpi_Elf object that owns the Ddpi_Type object for the type instance.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- type is NULL or invalid.
- ret_elf is NULL.

ddpi_type_get_die_offset operation

The ddpi_type_get_die_offset operation returns the DIE offset of the type in the .debug_info section.

Prototype

```
int ddpi_type_get_die_offset(
  Ddpi_Type             type,
  Dwarf_Off*            ret_offset,
  Ddpi_Error*           error);
```

Members

type
Input. This accepts the Ddpi_Type object for the instance.

ret_offset
Output. This returns the DIE offset for the type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `type` is NULL or invalid.
- `ret_offset` is NULL.

Chapter 22. Ddpi_Sourcefile APIs

Ddpi_Sourcefile APIs provide information about source files in the `.debug_srcfiles` section.

Ddpi_Sourcefile object

A Ddpi_Sourcefile object contains information about a source file. If this source file is in more than one compilation unit, there will be a separate Ddpi_Sourcefile object for each unit.

This object can be queried to get the following information:

- The full path name.
- The file name without the full path.
- The Ddpi_Access object whose `.dbg` file contains information from the source file.

Type definition

```
typedef struct Ddpi_Sourcefile_s* Ddpi_Sourcefile;
```

ddpi_sourcefile_get_full_name operation

The `ddpi_sourcefile_get_full_name` operation returns the full path name of the source file for the compilation unit. The actual version of the name is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_sourcefile_get_full_name(
  Ddpi_Sourcefile       sourcefile,
  char**                ret_full_name,
  Ddpi_Error*           error);
```

Parameters

sourcefile
Input. This accepts the `Ddpi_Sourcefile` object for the source file.

ret_full_name
Output. This returns the full path name of the source file.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `sourcefile` is NULL or invalid.
- The `Ddpi_Elf`, `Ddpi_Access`, or `Ddpi_Info` object associated with the source file is either NULL or invalid.

© Copyright IBM Corp. 2004, 2009

- ret_full_name is NULL.

ddpi_sourcefile_get_short_name operation

The ddpi_sourcefile_get_short_name operation returns the name of the source file for the compilation unit, without the full path. The actual version of the name is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_sourcefile_get_short_name(
  Ddpi_Sourcefile       sourcefile,
  char**                ret_short_name,
  Ddpi_Error*           error);
```

Parameters

sourcefile
Input. This accepts the Ddpi_Sourcefile object for the source file.

ret_short_name
Output. This returns the path name of the source file, without the path.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- sourcefile is NULL or invalid.
- The Ddpi_Elf, Ddpi_Access, or Ddpi_Info object associated with the source file is either NULL or invalid.
- ret_short_name is NULL.

ddpi_sourcefile_get_access operation

The ddpi_sourcefile_get_access operation returns the Ddpi_Access object whose Dwarf_Debug instance contains information from the source file for the compilation unit. The actual Ddpi_Access version is returned, not a copy. The user must never deallocate the returned pointer.

Prototype

```
int ddpi_sourcefile_get_access(
  Ddpi_Sourcefile       sourcefile,
  Ddpi_Access*          ret_access,
  Ddpi_Error*           error);
```

Parameters

sourcefile
Input. This accepts the Ddpi_Sourcefile object for instance.

ret_access
Output. This returns the `Ddpi_Access` object whose `Dwarf_Debug` object contains the instance information.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `sourcefile` is NULL or invalid.
- The `Ddpi_Elf`, `Ddpi_Access`, or `Ddpi_Info` object associated with the source file is either NULL or invalid.
- `ret_access` is NULL.

ddpi_sourcefile_get_die_offset operation

The `ddpi_sourcefile_get_die_offset` operation returns the DIE offset of the source file.

Prototype

```
int ddpi_sourcefile_get_die_offset(
   Ddpi_Sourcefile       sourcefile,
   Dwarf_Off*            ret_offset,
   Ddpi_Error*           error);
```

Members

sourcefile
Input. This accepts the `Ddpi_Sourcefile` object for the source file.

ret_offset
Output. This returns the DIE offset.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
This value is never returned.

DW_DLV_ERROR
This value is returned if:

- `sourcefile` is NULL or invalid.
- `ret_offset` is NULL.

ddpi_sourcefile_get_source_lines operation

The ddpi_sourcefile_get_source_lines operation returns the contents of the source file at the given line numbers. The returned string will be NULL terminated. To return all remaining lines, pass 0 for num_lines. If num_lines exceeds the remaining number of lines, all remaining lines will be returned.

Prototype

```
int ddpi_sourcefile_get_source_lines(
   Ddpi_Sourcefile       sourcefile,
   Dwarf_Unsigned        first_line,
   Dwarf_Unsigned        num_lines,
   char**                ret_source,
   Ddpi_Error*           error);
```

Parameters

sourcefile
Input. This accepts the Ddpi_Sourcefile object for the source file.

first_line
Input. This accepts the first line number to retrieve.

num_lines
Input. This accepts the number of lines to retrieve, or 0 to retrieve all remaining lines.

ret_source
Output. This returns the contents of the source file at the given lines.

error
Input or output. This accepts and returns the Ddpi_Error object. This is a required parameter that handles error information generated by the producer or consumer application. If error is not NULL, the error information will be stored in the given object. If error is NULL, the API will search the error handler that is specified by the ddpi_init function. If no handler is specified, the application will abort.

Return values

DW_DLV_OK
This value is returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
The given line numbers are out of range for the source file, or there is no captured source available for this file.

DW_DLV_ERROR
This value is returned if:

- sourcefile is NULL or invalid.
- The Ddpi_Elf, Ddpi_Access, Ddpi_Module, or Ddpi_Info object associated with sourcefile is either NULL or invalid.
- ret_source is NULL.

Chapter 23. Ddpi_EntryPt APIs

Ddpi_EntryPt contains the following information:

- The location of the entry point
- The PDS/PDSE member name with which the application-executable module was entered or could be entered
- The symbol or control-section (CSECT) name of the entry point
- The addressing mode (AMODE) of the entry point
- The location of the program code in memory that contains the entry point
- The offset from the low address to the actual entry point of the module

Ddpi_EntryPt object

The `Ddpi_EntryPt` object is an opaque data type that contains information regarding a single entry point for a given application-executable module.

Type definition

```
typedef struct Ddpi_EntryPt_s*    Ddpi_EntryPt;
```

Ddpi_EntryPt_Type object

Contains the entry-point type constants.

Type definition

```
typedef enum Ddpi_EntryPt_Type_s {
  Ddpi_EPT_Unknown         =  0,
  Ddpi_EPT_MVS_PDS_Member  =  1,
  Ddpi_EPT_MVS_PDS_Alias   =  2,
  Ddpi_EPT_Unnamed         =  3
} Ddpi_EntryPt_Type;
```

Members

Ddpi_EPT_Unknown
If this value is 0, the entry-point type is unknown or unacceptable.

Ddpi_EPT_MVS_PDS_Member
If this value is 1, the entry-point type is MVS or PDS.

Ddpi_EPT_MVS_PDS_Alias
If this value is 1, the entry-point type is an MVS or PDS alias.

Ddpi_EPT_Unnamed
If this value is 2, the entry-point type is unnamed.

ddpi_entrypt_create operation

The `ddpi_entrypt_create` operation creates an entry-point object to describe the entry point of a `Ddpi_Module` object and returns a descriptor.

When you call `ddpi_entrypt_create`, and pass a character string as the parameter name, the operation copies the content of `name` into its own storage. Once the operation returns the descriptor, you can deallocate the original name and save storage.

© Copyright IBM Corp. 2004, 2009

Prototype

```
int ddpi_entrypt_create(
  Ddpi_Module           module,
  char*                 entry_name,
  char*                 symbol_name,
  Ddpi_EntryPt_Type     type,
  Ddpi_Addr_Mode        mode,
  Ddpi_Class            storage,
  Dwarf_Off             storage_off,
  int                   user_area_len,
  Ddpi_EntryPt*         ret_entrypt,
  Ddpi_Error*           error);
```

Parameters

module
Input. This accepts the `Ddpi_Module` object.

entry_name
Input. This accepts the name or alias of the entry point. The value must be provided in ASCII.

symbol_name
Input. This accepts the entry point symbol name. For example, the value could be `CEESTART` for C/C++.

type
Input. This accepts the entry-point type.

mode
Input. This accepts the entry-point address mode.

storage
Input. This accepts the extent of the entry point storage.

storage_off
Input. This accepts the offset of the entry-point storage.

user_area_len
Input. This accepts the user-area length.

ret_entrypt
Output. This returns a descriptor, which points to the entry-point object just created.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the entry-point object descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `mode` is NULL.
- The Ddpi_Info object associated with `module` is NULL.
- `symbol_name` is NULL.
- `ret_entrypt` is NULL.
- `storage` the (Ddpi_Class) is NULL.

- An error occurs during memory allocation.
- `user_area_len` is less than zero.

ddpi_entrypt_term operation

The `ddpi_entrypt_term` operation terminates an entry-point object.

The `ddpi_entrypt_term` operation releases all internal resources associated with the entry-point descriptor and invalidates the entry point.

Prototype

```
int ddpi_entrypt_term(
  Ddpi_EntryPt          entrypt,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the entry-point object descriptor.

DW_DLV_NO_ENTRY
Returned if the entry point was not found in its parent’s list of modules.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The Ddpi_Module or Ddpi_Info object associated with `entrypt` is NULL.
- An error occurs while terminating child descriptors.
- An error occurs during memory allocation.

ddpi_entrypt_get_owner operation

The `ddpi_entrypt_get_owner` operation returns the owner of an entry-point object, and returns a pointer to the owner.

Prototype

```
int ddpi_entrypt_get_owner(
  Ddpi_EntryPt          entrypt,
  Ddpi_Module*          ret_owner,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

ret_owner
Output. This returns the `Ddpi_Module` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the entry-point object pointer.

DW_DLV_NO_ENTRY
Returned if the entry point was not found in its parent's list of modules.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The `Ddpi_Info` object associated with `entrypt` is NULL.
- `ret_owner` is NULL

ddpi_entrypt_get_entry_name operation

The `ddpi_entrypt_get_entry_name` operation returns the entry (or alias) name of an entry-point object, and returns a pointer to the name.

The name string is part of the `Ddpi_EntryPt` object, and must not be deallocated.

Prototype

```
int ddpi_entrypt_get_entry_name(
  Ddpi_EntryPt          entrypt,
  char**                ret_entry_name,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

ret_entry_name
Output. This returns a pointer to the entry name.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the entry-point alias name.

DW_DLV_NO_ENTRY
Returned if the entry name is not available.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The `Ddpi_Info` object associated with `entrypt` is NULL.
- `ret_entry_name` is NULL

ddpi_entrypt_set_entry_name operation

The `ddpi_entrypt_set_entry_name` operation assigns a new entry (or alias) name of an entry-point object.

When the user calls the ddpi_entrypt_set_entry_name, passing a character string as the parameter name, the operation copies the content of name into its own storage. After the operation returns a value, you can deallocate the original name and save storage.

Prototype

```
int ddpi_entrypt_set_entry_name(
  Ddpi_EntryPt          entrypt,
  char*                 new_entry_name,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

new_entry_name
Input. This accepts a new entry name.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the entry-point alias name.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- entrypt is NULL.
- The Ddpi_Info object associated with entrypt is NULL.

ddpi_entrypt_get_symbol_name operation

The ddpi_entrypt_get_symbol_name operation returns the symbol name of an entry-point object.

The name string is part of the Ddpi_EntryPt object, and must not be deallocated.

Prototype

```
int ddpi_entrypt_get_symbol_name(
  Ddpi_EntryPt          entrypt,
  char**                ret_symbol_name,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

ret_symbol_name
Output. This returns the symbol name of the entry point.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the entry-point alias name.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The Ddpi_Info object associated with `entrypt` is NULL.
- `ret_symbol_name` is NULL.

ddpi_entrypt_set_symbol_name operation

The `ddpi_entrypt_set_symbol_name` operation assigns a new symbol name of an entry-point object.

When the user calls `ddpi_entrypt_set_symbol_name`, passing a character string as the parameter name, the operation copies the content of `name` into its own storage. After the operation returns a value, you can deallocate the original name and save storage.

Prototype

```
int ddpi_entrypt_set_symbol_name(
  Ddpi_EntryPt          entrypt,
  char*                 new_symbol_name,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

new_symbol_name
Input. This accepts the new symbol name of the entry point.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the new symbol name.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The `Ddpi_Info` object associated with `entrypt` is NULL.
- `new_symbol_name` is NULL or 0 bytes long

ddpi_entrypt_get_type operation

The `ddpi_entrypt_get_type` operation returns the type of an entry-point object.

Prototype

```
int ddpi_entrypt_get_type(
  Ddpi_EntryPt          entrypt,
  Ddpi_EntryPt_Type*    ret_type,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

ret_type
Output. This returns the type of the entry point.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the entry-point object type.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The `Ddpi_Info` object associated with `entrypt` is NULL.
- `ret_type` is NULL

ddpi_entrypt_set_type operation

The `ddpi_entrypt_set_type` operation assigns a new type of an entry-point object.

Prototype

```
int ddpi_entrypt_set_type(
  Ddpi_EntryPt          entrypt,
  Ddpi_EntryPt_Type     new_type,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

ret_type
Input. This accepts the new type of the entry point.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the entry-point object type.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The `Ddpi_Info` object associated with `entrypt` is NULL.
- `ret_type` is NULL

ddpi_entrypt_get_addr_mode operation

The `ddpi_entrypt_get_addr_mode` operation returns the address mode (AMODE) of an entry-point object.

Prototype

```
int ddpi_entrypt_get_addr_mode(
  Ddpi_EntryPt          entrypt,
  Ddpi_Addr_Mode*       ret_amode,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

ret_amode
Output. This returns the addressing mode of the entry point.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the entry-point address mode.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The `Ddpi_Info` object associated with `entrypt` is NULL.
- `ret_amode` is NULL.

ddpi_entrypt_set_addr_mode operation

The `ddpi_entrypt_set_addr_mode` operation assigns a new AMODE of an entry-point object.

Prototype

```
int ddpi_entrypt_set_addr_mode(
  Ddpi_EntryPt          entrypt,
  Ddpi_Addr_Mode        new_amode,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object (`Ddpi_EntryPt`).

new_amode
Input. This accepts the object that contains the new entry-point AMODE (`Ddpi_Addr_Mode`).

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the entry-point address mode.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The `Ddpi_Info` object associated with `entrypt` is NULL.

ddpi_entrypt_get_storage_extent operation

The `ddpi_entrypt_get_storage_extent` operation returns a pointer to the storage-extent class of an entry-point object.

Do not terminate or delete this pointer. It will be terminated when `libddpi` is terminated.

Prototype

```
int ddpi_entrypt_get_storage_extent(
  Ddpi_EntryPt          entrypt,
  Ddpi_Class*           ret_storage_extent,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

ret_storage_extent
Output. This returns the `Ddpi_Class` object of the entry point.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the pointer to the storage-extent class of the entry-point object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The `Ddpi_Info` object associated with `entrypt` is NULL.
- `ret_storage_extent` is NULL.

ddpi_entrypt_set_storage_extent operation

The ddpi_entrypt_set_storage_extent operation assigns the storage extent of an entry-point object to a Ddpi_Class object.

Do not terminate or delete this pointer. It will be terminated when libddpi is terminated.

Prototype

```
int ddpi_entrypt_set_storage_extent(
  Ddpi_EntryPt          entrypt,
  Ddpi_Class            new_storage_extent,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

new_storage_extent
Input. This accepts the new Ddpi_Class object of the entry point.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of a pointer to the storage-extent class of the entry-point object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- entrypt is NULL.
- The Ddpi_Info object associated with entrypt is NULL.
- new_storage_extent is NULL.

ddpi_entrypt_get_storage_offset operation

The ddpi_entrypt_get_storage_offset operation returns the storage offset of an entry-point object.

The offset is relative to the start address of the storage-extent Ddpi_Class object.

Prototype

```
int ddpi_entrypt_get_storage_offset(
  Ddpi_EntryPt          entrypt,
  Dwarf_Off*            ret_storage_off,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

ret_storage_off
Output. This returns the storage offset of the entry point.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the entry-point storage offset.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The `Ddpi_Info` object associated with `entrypt` is NULL.
- `ret_storage_extent` is NULL.

ddpi_entrypt_set_storage_offset operation

The `ddpi_entrypt_set_storage_offset` operation assigns the storage offset of an entry-point object.

The offset is relative to the start address of the storage-extent `Ddpi_Class` object.

Prototype

```
int ddpi_entrypt_set_storage_offset(
  Ddpi_EntryPt          entrypt,
  Dwarf_Off             new_storage_off,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

new_storage_off
Input. This accepts the storage offset of the entry point.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the entry-point storage offset.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- The `Ddpi_Info` object associated with `entrypt` is NULL.

ddpi_entrypt_get_user_area operation

The `ddpi_entrypt_get_user_area` operation queries the user area of an entry-point object. It then returns a pointer to the start of the user area.

Prototype

```
int ddpi_entrypt_get_user_area(
  Ddpi_EntryPt          entrypt,
  Dwarf_Ptr*            ret_user_area,
  Ddpi_Error*           error);
```

Parameters

entrypt
Input. This accepts the entry-point object.

ret_user_area
Output. This returns a pointer to the user area of the entry point.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the user area start address.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `entrypt` is NULL.
- `ret_user_area` is NULL

Chapter 24. Ddpi_Machinestate APIs

This object contains members that represent the different registers and other machine-related information, such as the program status word (PSW). A `Ddpi_Machinestate` object can represent either the actual (current) machine state or the model (original) machine state.

Use the `Ddpi_Machinestate` object to show the current state of the system and its registers.

In order to create a model, the registers themselves, not the objects, are changed to represent the modeled state. An application uses the `Ddpi_Machinestate` object to keep track of the original state of the machine. After it stops actively debugging an executable module, the debugger can return the machine to its original state.

Note: `Ddpi_Machinestate` is also used by the formatting operations. For more information, see Chapter 30, “Ddpi_Format APIs,” on page 295.

`Ddpi_Machinestate` objects must be created by the `ddpi_machinestate_create` operation and initialized by the `ddpi_machinestate_init` operation.

Use set and get operations to assign and verify values.

Use query operations to determine if the value has been modified since the last time all of the modified bits were reset by the `ddpi_machinestate_reset_change` operation.

Use context APIs to relate a machine state with a currently-operating task.

The PSW contains both the instruction pointer (IP) address and the addressing mode (AMODE). The Ddpi_Machinestate object allows the IP address and AMODE to be updated separately from the PSW. Any change in IP or AMODE can affect the validity of the PSW, and a change in PSW affects the value and validity of the IP and AMODE.

Ddpi_MachineState object

The `Ddpi_MachineState` object is an opaque data structure that associates all information regarding the given machine state for a given context.

Type definition

```
typedef struct Ddpi_MachineState_s*   Ddpi_MachineState;
```

Ddpi_AR object

The `Ddpi_AR` object contains valid address register (AR) types.

Type definition

```
typedef enum Ddpi_AR_s {
   Ddpi_AR00   =  0,
   Ddpi_AR01   =  1,
   Ddpi_AR02   =  2,
   Ddpi_AR03   =  3,
   Ddpi_AR04   =  4,
   Ddpi_AR05   =  5,
```

© Copyright IBM Corp. 2004, 2009

```
   Ddpi_AR06   =  6,
   Ddpi_AR07   =  7,
   Ddpi_AR08   =  8,
   Ddpi_AR09   =  9,
   Ddpi_AR10   = 10,
   Ddpi_AR11   = 11,
   Ddpi_AR12   = 12,
   Ddpi_AR13   = 13,
   Ddpi_AR14   = 14,
   Ddpi_AR15   = 15
 } Ddpi_AR;
```

Members

Ddpi_AR00
0

Ddpi_AR01
1

Ddpi_AR02
2

Ddpi_AR03
3

Ddpi_AR04
4

Ddpi_AR05
5

Ddpi_AR06
6

Ddpi_AR07
7

Ddpi_AR08
8

Ddpi_AR09
9

Ddpi_AR10
10

Ddpi_AR11
11

Ddpi_AR12
12

Ddpi_AR13
13

Ddpi_AR14
14

Ddpi_AR15
15

Ddpi_CR object

The `Ddpi_CR` object contains valid control register (CR) types.

Type definition

```
typedef enum Ddpi_CR_s {
   Ddpi_CR00   =  0,
   Ddpi_CR01   =  1,
   Ddpi_CR02   =  2,
   Ddpi_CR03   =  3,
   Ddpi_CR04   =  4,
   Ddpi_CR05   =  5,
   Ddpi_CR06   =  6,
   Ddpi_CR07   =  7,
   Ddpi_CR08   =  8,
   Ddpi_CR09   =  9,
   Ddpi_CR10   = 10,
   Ddpi_CR11   = 11,
   Ddpi_CR12   = 12,
   Ddpi_CR13   = 13,
   Ddpi_CR14   = 14,
   Ddpi_CR15   = 15
 } Ddpi_CR;
```

Members

Ddpi_CR00
0

Ddpi_CR01
1

Ddpi_CR02
2

Ddpi_CR03
3

Ddpi_CR04
4

Ddpi_CR05
5

Ddpi_CR06
6

Ddpi_CR07
7

Ddpi_CR08
8

Ddpi_CR09
9

Ddpi_CR10
10

Ddpi_CR11
11

Ddpi_CR12
12

Ddpi_CR13
13

Ddpi_CR14
14

Ddpi_CR15
15

Ddpi_FPR object

The Ddpi_FPR object contains valid floating-point register (FPR) types.

Type definition

```
typedef enum Ddpi_FPR_s {
   Ddpi_FPR00   =  0,
   Ddpi_FPR01   =  1,
   Ddpi_FPR02   =  2,
   Ddpi_FPR03   =  3,
   Ddpi_FPR04   =  4,
   Ddpi_FPR05   =  5,
   Ddpi_FPR06   =  6,
   Ddpi_FPR07   =  7,
   Ddpi_FPR08   =  8,
   Ddpi_FPR09   =  9,
   Ddpi_FPR10   = 10,
   Ddpi_FPR11   = 11,
   Ddpi_FPR12   = 12,
   Ddpi_FPR13   = 13,
   Ddpi_FPR14   = 14,
   Ddpi_FPR15   = 15
 } Ddpi_FPR;
```

Members

Ddpi_FPR00
0

Ddpi_FPR01
1

Ddpi_FPR02
2

Ddpi_FPR03
3

Ddpi_FPR04
4

Ddpi_FPR05
5

Ddpi_FPR06
6

Ddpi_FPR07
7

Ddpi_FPR08
8

Ddpi_FPR09
9

Ddpi_FPR10
10

Ddpi_FPR11
11

Ddpi_FPR12
12

Ddpi_FPR13
13

Ddpi_FPR14
14

Ddpi_FPR15
15

Ddpi_GPR object

The `Ddpi_GPR` object contains valid general-purpose register (GPR) types.

Type definition

```
typedef enum Ddpi_GPR_s {
   Ddpi_GPR00   =  0,
   Ddpi_GPR01   =  1,
   Ddpi_GPR02   =  2,
   Ddpi_GPR03   =  3,
   Ddpi_GPR04   =  4,
   Ddpi_GPR05   =  5,
   Ddpi_GPR06   =  6,
   Ddpi_GPR07   =  7,
   Ddpi_GPR08   =  8,
   Ddpi_GPR09   =  9,
   Ddpi_GPR10   = 10,
   Ddpi_GPR11   = 11,
   Ddpi_GPR12   = 12,
   Ddpi_GPR13   = 13,
   Ddpi_GPR14   = 14,
   Ddpi_GPR15   = 15
 } Ddpi_GPR;
```

Members

Ddpi_GPR00
0

Ddpi_GPR01
1

Ddpi_GPR02
2

Ddpi_GPR03
3

Ddpi_GPR04
4

Ddpi_GPR05
5

Ddpi_GPR06
6

Ddpi_GPR07
7

Ddpi_GPR08
8

Ddpi_GPR09
9

Ddpi_GPR10
10

Ddpi_GPR11
11

Ddpi_GPR12
12

Ddpi_GPR13
13

Ddpi_GPR14
14

Ddpi_GPR15
15

Ddpi_PSW object

The Ddpi_PSW object contains valid program status word (PSW) types.

Type definition

```
typedef struct Ddpi_PSW_s {
   int        zero_field0:1;
   int        per_mask:1;
   int        zero_field1:2;
   int        bit4:1;
   int        dat_mode:1;
   int        io_mask:1;
   int        external_mask:1;
   int        psw_key:4;
   int        bit12:1;
   int        machine_check_mask:1;
   int        wait_state:1;
   int        problem_state:1;
   int        as_control:2;
   int        condition_code:2;
   int        program_mask:4;
   int        zero_field2:7;
   int        extended_am:1;
   int        basic_am:1;
   int        zero_field3:31;
   Dwarf_Addr IP;
} Ddpi_PSW;
```

Members

zero_field0:1
Integer type. Reserved. The field must be zero.

per_mask:1
Integer type. Program-event recording mask.

zero_field1:2
Integer type. Reserved. The field must be zero.

bit4:1
Integer type. Bit 4.

dat_mode:1
Integer type. Controls the dynamic address translation (DAT) of storage addresses during storage access.

io_mask:1
Integer type. I/O interrupt mask.

external_mask:1
Integer type. External interrupt mask.

psw_key:4
Integer type. Storage-access key.

bit12:1
Integer type. Bit 12 must be 0 for the MVS-architecture PSW.

machine_check_mask:1
Integer type. Machine-check interrupt mask.

wait_state:1
Integer type. 1. No instructions processed by the CPU, but interruptions may take place. 0. Normal instruction fetch and execution.

problem_state:1
Integer type. 1. CPU in problem state. 0. CPU in supervisor state.

as_control:2
Integer type. Address-translation mode control.

condition_code:2
Integer type. Condition code.

program_mask:4
Integer type. Program mask bits, one per exception type

zero_field2:7
Integer type. Reserved. The field must be zero.

extended_am:1
Integer type. Controls size of effective addresses, in conjunction with the `basic_am` .

basic_am:1
Integer type. Basic-address mode control.

zero_field3:31
Integer type. Reserved. The field must be zero.

IP `Dwarf_Addr` type. Instruction-address pointer.

Ddpi_PSW_Type object

The `Ddpi_PSW_Type` object contains the current PSW type.

Type definition

```
typedef enum Ddpi_PSW_Type_s {
   Ddpi_PSW_T_current    = 0,
   Ddpi_PSW_T_machchk    = 1,
   Ddpi_PSW_T_io         = 2
} Ddpi_PSW_Type;
```

Members

Ddpi_PSW_T_current
If this value is 0, the PSW gives the execution status.

Ddpi_PSW_T_machchk
If this value is 1, the PSW gives the machine state.

Ddpi_PSW_T_io
If this value is 2, the PSW gives the I/O state.

Ddpi_Context object

The `Ddpi_Context` object contains valid context data types.

Type definition

```
typedef union Ddpi_Context_s {
  Ddpi_Thread           cx_thread;
  Ddpi_Process          cx_process;
} Ddpi_Context;
```

Members

cx_thread
`Ddpi_Thread` type with pthread context information.

cx_process
`Ddpi_Process` type with MVS TCB context information.

Ddpi_Context_Type object

The `Ddpi_Context_Type` object contains the data type of the context in which the compilation unit is running.

Type definition

```
typedef enum Ddpi_Context_Type_s {
  Ddpi_CX_unknown       =  0,
  Ddpi_CX_pthread       =  1,
  Ddpi_CX_TCB           =  2
} Ddpi_Context_Type;
```

Members

Ddpi_CX_unknown
If this value is 0, the context data type is unknown.

Ddpi_CX_pthread
If this value is 1, the program is running under a POSIX thread.

Ddpi_CX_TCB
If this value is 2, the program is running under MVS TCB.

ddpi_machinestate_create operation

The `ddpi_machinestate_create` operation creates a `Ddpi_MachineState` object to describe the processor state and returns a descriptor that represents the system state at the time the descriptor was created.

Prototype

```
int ddpi_machinestate_create(
  Ddpi_Space            space,
  Ddpi_Context_Type     context_type,
  Ddpi_Context          context,
  Ddpi_MachineState*    ret_machinestate,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

context_type
Input. This accepts the execution context type.

context
Input. This accepts the `Ddpi_Context` object.

ret_machinestate
Output. This returns the `Ddpi_MachineState` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the machine state object descriptor.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL.
- The `Ddpi_Info` object associated with `space` is NULL.
- `ret_machinestate` is NULL.
- An error occurs during memory allocation.

ddpi_machinestate_term operation

The `ddpi_machinestate_term` operation terminates the given `Ddpi_MachineState` object.

`ddpi_machinestate_term` releases all internal resources associated with the machine state object, and invalidates the given machine state.

Prototype

```
int ddpi_machinestate_term(
  Ddpi_MachineState     machinestate,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful invalidation of the given machine state object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL.
- The `Ddpi_Space` or `Ddpi_Info` object associated with `space` is NULL.
- `ret_machinestate` is NULL.
- A problem occurs during termination of the stack state.
- An error occurs during memory allocation.

ddpi_machinestate_init operation

The `ddpi_machinestate_init` operation resets the given `Ddpi_MachineState` object and its associated `Ddpi_StorageLocn` object.

`ddpi_machinestate_init` never attempts to reset the `Ddpi_Space` object and the `Ddpi_Context` object.

The nested `Ddpi_StorageLocn` is set back to `SL_Policy_Opaque` and the `address` is set back to 0.

Prototype

```
int ddpi_machinestate_init(
  Ddpi_MachineState     machinestate,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful invalidation of the given machine state object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- `ddpi_storagelocn_set_policy` returns DW_DLV_ERROR.
- `ddpi_storagelocn_set_addr` returns DW_DLV_ERROR.

ddpi_machinestate_clone operation

The ddpi_machinestate_clone operation clones the internal state of a Ddpi_MachineState object to a new Ddpi_MachineState object.

The ddpi_machinestate_clone operation creates the new Ddpi_MachineState object by:

1. Creating a Ddpi_MachineState object.
2. Copying the states from the given Ddpi_MachineState object into the new Ddpi_MachineState object.
3. Creating a new Ddpi_StorageLocn object with the same settings as the Ddpi_StorageLocn object from the original Ddpi_MachineState object.

Prototype

```
int ddpi_machinestate_clone(
  Ddpi_MachineState     original,
  Ddpi_MachineState*    clone_ptr,
  Ddpi_Error*           error);
```

Parameters

original
Input. This accepts the original Ddpi_MachineState object.

clone_ptr
Output. This returns a pointer to the Ddpi_MachineState object that will receive the clone information.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful creation of the given machine state object clone.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- original is NULL
- The Ddpi_Space or Ddpi_Info object associated with original is NULL.
- An error occurs during creation of the nested Ddpi_StorageLocn object.

ddpi_machinestate_copy operation

The ddpi_machinestate_copy operation copies the internal state of a Ddpi_MachineState object to another existing Ddpi_MachineState object.

The ddpi_machinestate_copy operation creates a new Ddpi_StorageLocn with the same Ddpi_StorageLocn settings as the original Ddpi_MachineState.

Prototype

```
int ddpi_machinestate_copy(
  Ddpi_MachineState     original,
  Ddpi_MachineState     copy,
  Ddpi_Error*           error);
```

Parameters

original
Input. This accepts the original `Ddpi_MachineState` object.

copy
Input. This accepts the `Ddpi_MachineState` object to be updated with the copy.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful creation of the storage location copy.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `original` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `original` is NULL.
- `copy` is NULL
- An error occurs during update or access of the nested `Ddpi_StorageLocn` object.

ddpi_machinestate_get_gpr operation

The `ddpi_machinestate_get_gpr` operation returns specific GPR elements of the `Ddpi_MachineState` object.

`ddpi_machinestate_copy` returns:

- The GPR value
- The GPR high and low flags for the given GPR, depending on settings given at the time of the flags are assigned
- The GPR value modified flag, to indicate whether the GPR value has been modified since the last reset
- The stored location of the GPR, which may be used when shadowing GPR values or walking a stack

Prototype

```
int ddpi_machinestate_get_gpr(
  Ddpi_GPR              gpr_nbr,
  Ddpi_MachineState     machinestate,
  Dwarf_Signed *        ret_gpr_value,
  Dwarf_Bool*           ret_gpr_high_valid,
  Dwarf_Bool*           ret_gpr_low_valid,
  Dwarf_Bool*           ret_gpr_modified,
  Dwarf_Unsigned *      ret_gpr_locn,
  Ddpi_Error*           error);
```

Parameters

gpr_nbr
Input. This accepts the GPR number.

machinestate
Input. This accepts the original `Ddpi_MachineState` object.

ret_gpr_value
Output. This returns the GPR value.

ret_gpr_high_valid
Output. This returns the high GPR valid flag.

ret_gpr_low_valid
Output. This returns the low GPR valid flag.

ret_gpr_modified
Output. This returns the GPR modified flag.

ret_gpr_locn
Output. This returns the GPR location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieva of the GPR settings.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- `gpr_nbr` is out of range
- `ret_gpr_value`, `ret_gpr_high_valid`, `ret_gpr_low_valid`, `ret_gpr_modified`, or `ret_gpr_locn` are NULL
- An error occurs during creation of the nested `Ddpi_StorageLocn` object.
- An error occurs when a DWARF function is called.

ddpi_machinestate_set_gpr operation

The `ddpi_machinestate_set_gpr` operation updates the GPR value, flags, and location of a given `Ddpi_MachineState` object.

Prototype

```
int ddpi_machinestate_set_gpr(
  Ddpi_GPR              gpr_nbr,
  Ddpi_MachineState     machinestate,
  Dwarf_Signed          gpr_value,
  Dwarf_Bool            gpr_high_valid,
  Dwarf_Bool            gpr_low_valid,
  Dwarf_Unsigned        gpr_locn,
  Ddpi_Error*           error);
```

Parameters

gpr_nbr
Input. This accepts the GPR number.

machinestate
Input. This accepts the `Ddpi_MachineState` object.

gpr_value
Input. This accepts the new GPR value.

gpr_high_valid
Input. This accepts the new high GPR valid flag.

gpr_low_valid
Input. This accepts the new low GPR valid flag.

gpr_locn
Input. This accepts the new GPR location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the saved GPR settings to the new machine state object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- `gpr_nbr` is out of range.
- An unknown value is given for the valid flags.
- An error occurs when a DWARF function is called.

ddpi_machinestate_query_gpr_change operation

The `ddpi_machinestate_query_gpr_change` operation checks the state of a given GPR for any change, and returns true if a bit setting has been modified.

Prototype

```
int ddpi_machinestate_query_gpr_change(
  Ddpi_MachineState     machinestate,
  Dwarf_Bool*           ret_flag,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_flag
Output. This returns true if the state of the GPR has changed.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the flag that indicates whether a change in GPR has occurred.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- An error occurs when a DWARF function is called.

ddpi_machinestate_get_fpr operation

The `ddpi_machinestate_get_fpr` operation returns specific FPR values of the given `Ddpi_MachineState` object.

The `ddpi_machinestate_get_fpr` operation returns:

- The value of the requested FPR number.
- The valid flag for the given FPR. This is dependent on the last value given by the user.
- The modified flag for the given FPR. This indicates whether the FPR value has been modified since the last reset.
- The stored location of this FPR. This may be used when shadowing FPR values.

Prototype

```
int ddpi_machinestate_get_fpr(
  Ddpi_FPR              fpr_nbr,
  Ddpi_MachineState     machinestate,
  Dwarf_Signed*         ret_fpr_value,
  Dwarf_Bool*           ret_fpr_valid,
  Dwarf_Bool*           ret_fpr_modified,
  Dwarf_Unsigned*       ret_fpr_locn,
  Ddpi_Error*           error);
```

Parameters

fpr_nbr
Input. This accepts the FPR number.

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_fpr_value
Output. This returns the FPR value.

ret_fpr_valid
Output. This returns the FPR valid flag.

ret_fpr_modified
Output. This returns the FPR modified flag.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the FPR settings.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- machinestate is NULL
- The Ddpi_Space or Ddpi_Info object associated with machinestate is NULL.
- ret_fpr_value, ret_fpr_valid, ret_fpr_modified, or ret_fpr_locn is NULL.
- fpr_nbr is out of range.
- An error occurs when a DWARF function is called.

ddpi_machinestate_set_fpr operation

The ddpi_machinestate_set_fpr operation updates the given FPR of a given Ddpi_MachineState object.

The ddpi_machinestate_set_fpr operation updates the FPR value, flag, and location.

Prototype

```
int ddpi_machinestate_set_fpr(
  Ddpi_FPR              fpr_nbr,
  Ddpi_MachineState     machinestate,
  Dwarf_Signed          fpr_value,
  Dwarf_Bool            fpr_valid,
  Dwarf_Unsigned        fpr_locn,
  Ddpi_Error*           error);
```

Parameters

fpr_nbr
Input. This accepts the FPR number.

machinestate
Input. This accepts the Ddpi_MachineState object.

fpr_value
Input. This accepts the FPR value.

fpr_valid
Input. This accepts the FPR valid flag.

fpr_locn
Input. This accepts the new FPR location, or 0.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful update of the FPR settings.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- machinestate is NULL.
- The Ddpi_Space or Ddpi_Info object associated with machinestate is NULL.
- ret_fpr_value, ret_fpr_valid, ret_fpr_modified, or ret_fpr_locn is NULL.
- fpr_nbr is out of range.
- An error occurs when a DWARF function is called.

ddpi_machinestate_query_fpr_change operation

The ddpi_machinestate_query_fpr_change operation returns true if a bit setting on any FPR is modified.

Prototype

```
int ddpi_machinestate_query_fpr_change(
  Ddpi_MachineState     machinestate,
  Dwarf_Bool*           ret_flag,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the Ddpi_MachineState object.

ret_flag
Output. This returns true if the state of the FPR has changed.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the flag that indicates whether a change in FPR has occurred.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- machinestate is NULL
- The Ddpi_Space or Ddpi_Info object associated with machinestate is NULL.
- An error occurs when a DWARF function is called.

ddpi_machinestate_get_fpcr operation

The ddpi_machinestate_get_fpcr operation returns specific floating-point control register (FPCR) elements of the given Ddpi_MachineState object.

ddpi_machinestate_get_fpcr returns the following elements:

- The FPCR value of the given Ddpi_MachineState object
- The valid flag for this FPCR, based on the last value given when setting the FPCR
- The modified flag for this FPCR, which indicates whether the FPCR value has been modified since the last reset
- The stored location of this FPCR, which may be used when shadowing FPCR values

Prototype

```
int ddpi_machinestate_get_fpcr(
  Ddpi_MachineState     machinestate,
  Dwarf_Signed*         ret_fpcr_value,
```

```
  Dwarf_Bool*           ret_fpcr_valid,
  Dwarf_Bool*           ret_fpcr_modified,
  Dwarf_Unsigned*       ret_fpcr_locn,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_fpcr_value
Output. This returns the FCPR value.

ret_fpcr_valid
Output. This returns the FCPR valid flag.

ret_fpcr_modified
Output. This returns the FCPR modified flag.

ret_fpcr_locn
Output. This returns the FCPR location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the FPCR settings.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- `ret_fpcr_value`, `ret_fpcr_valid`, `ret_fpcr_modified`, or `ret_fpcr_locn` is NULL.
- An error occurs when a DWARF function is called.

ddpi_machinestate_set_fpcr operation

The `ddpi_machinestate_set_fpcr` operation updates specific FPCR elements of the given `Ddpi_MachineState` object.

`ddpi_machinestate_set_fpcr` updates the following elements:

- The FPCR value of the given `Ddpi_MachineState` object
- The valid flag for this FPCR, based on the last value given when setting the FPCR
- The stored location of this FPCR, which may be used when shadowing FPCR values

Prototype

```
int ddpi_machinestate_set_fpcr(
  Ddpi_MachineState     machinestate,
  Dwarf_Signed          fpcr_value,
  Dwarf_Bool            fpcr_valid,
  Dwarf_Unsigned        fpcr_locn,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

fpcr_value
Input. This accepts the FPCR value.

fpcr_valid
Input. This accepts the FPCR valid flag.

fpcr_locn
Input. This accepts the new FPCR location, or 0.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the FPCR settings.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- An unknown value is given for the valid flag
- An error occurs when a DWARF function is called.

ddpi_machinestate_query_fpcr_change operation

The `ddpi_machinestate_query_fpcr_change` operation queries a given `Ddpi_MachineState` object, and returns true if a bit setting has been modified on any FPCR element.

Prototype

```
int ddpi_machinestate_query_fpcr_change(
  Ddpi_MachineState     machinestate,
  Dwarf_Bool*           ret_flag,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_flag
Output. This returns true if the state of the FPCR has changed.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the flag that indicates whether a change in FPCR has occurred.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- An error occurs when a DWARF function is called.

ddpi_machinestate_get_ar operation

The `ddpi_machinestate_get_ar` operation returns specific AR elements of the given `Ddpi_MachineState` object.

`ddpi_machinestate_get_ar` returns the following elements:

- The value of the requested AR number
- The valid flag for the given AR, based on the last value given when setting the AR
- The modified flag for the given AR, which indicates whether the AR value has been modified since the last reset
- The stored location of this given AR, which may be used when shadowing AR values

Prototype

```
int ddpi_machinestate_get_ar(
  Ddpi_AR               ar_nbr,
  Ddpi_MachineState     machinestate,
  Dwarf_Signed*         ret_ar_value,
  Dwarf_Bool*           ret_ar_valid,
  Dwarf_Bool*           ret_ar_modified,
  Dwarf_Unsigned*       ret_ar_locn,
  Ddpi_Error*           error);
```

Parameters

ar_nbr
Input. This accepts the AR number.

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_ar_value
Output. This returns the AR value.

ret_ar_valid
Output. This returns the AR valid flag.

ret_ar_modified
Output. This returns the AR modified flag.

ret_ar_locn
Output. This returns the AR location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the AR settings.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- `ret_ar_value`, `ret_ar_valid`, `ret_ar_modified`, or `ret_ar_locn` is NULL.
- `ar_number` is out of range.
- An error occurs when a DWARF function is called.

ddpi_machinestate_set_ar operation

The `ddpi_machinestate_set_ar` operation updates the given AR of the given `Ddpi_MachineState` object.

`ddpi_machinestate_set_ar` updates the AR value, flag, and location.

Prototype

```
int ddpi_machinestate_set_ar(
  Ddpi_AR              ar_nbr,
  Ddpi_MachineState    machinestate,
  Dwarf_Signed         ar_value,
  Dwarf_Bool           ar_valid,
  Dwarf_Unsigned       ar_locn,
  Ddpi_Error*          error);
```

Parameters

ar_nbr
Input. This accepts the AR number.

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ar_value
Input. This accepts the new AR value list.

ar_valid
Input. This accepts the new AR valid flag.

ar_locn
Input. This accepts the new AR location, or 0.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the AR settings to the given machine state object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.

- ar_nbr is out of range
- An unknown value is given for the valid flag
- An error occurs when a DWARF function is called.

ddpi_machinestate_query_ar_change operation

The ddpi_machinestate_query_ar_change operation queries the given Ddpi_MachineState object and returns true if any bit setting has been modified any AR.

Prototype

```
int ddpi_machinestate_query_ar_change
  Ddpi_MachineState     machinestate,
  Dwarf_Bool*           ret_flag,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the Ddpi_MachineState object.

ret_flag
Output. This returns true if the state of the AR has changed.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the flag that indicates whether a change in AR has occurred.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- machinestate is NULL
- The Ddpi_Space or Ddpi_Info object associated with machinestate is NULL.
- An error occurs when a DWARF function is called.

ddpi_machinestate_get_cr operation

The ddpi_machinestate_get_cr operation returns specific CR elements of the given Ddpi_MachineState object.

ddpi_machinestate_get_cr returns the following elements:

- The value of the requested CR number
- The valid flag for the given CR, based on the last value given when setting the CR
- The modified flag for the given CR, which indicates whether the CR value has been modified since the last reset
- The stored location of this given CR, which may be used when shadowing CR values

Prototype

```
int ddpi_machinestate_get_cr(
  Ddpi_CR               cr_nbr,
  Ddpi_MachineState     machinestate,
  Dwarf_Signed*         ret_cr_value,
  Dwarf_Bool*           ret_cr_valid,
  Dwarf_Bool*           ret_cr_modified,
  Dwarf_Unsigned*       ret_cr_locn,
  Ddpi_Error*           error);
```

Parameters

cr_nbr
Input. This accepts the CR number.

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_cr_value
Output. This returns the CR value.

ret_cr_valid
Output. This returns the CR valid flag.

ret_cr_modified
Output. This returns the CR modified flag.

ret_cr_locn
Output. This returns the CR location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the CR settings.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- `ret_cr_value`, `ret_cr_valid`, `ret_cr_modified`, or `ret_cr_locn` is NULL.
- `cr_number` is out of range.
- An error occurs when a DWARF function is called.

ddpi_machinestate_set_cr operation

The `ddpi_machinestate_set_cr` operation updates the given CR of a given `Ddpi_MachineState` object.

`ddpi_machinestate_set_cr` updates the CR value, flag, and location.

Prototype

```
int ddpi_machinestate_set_cr(
  Ddpi_CR               cr_nbr,
  Ddpi_MachineState     machinestate,
```

```
  Dwarf_Signed          cr_value,
  Dwarf_Bool            cr_valid,
  Dwarf_Unsigned        cr_locn,
  Ddpi_Error*           error);
```

Parameters

cr_nbr
Input. This accepts the CR number.

machinestate
Input. This accepts the `Ddpi_MachineState` object.

cr_value
Input. This accepts the new CR value list.

cr_valid
Input. This accepts the new CR valid flag.

cr_locn
Input. This accepts the new CR location, or 0.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful update of the CR settings.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- `cr_nbr` is out of range.
- An unknown value is given for the valid flag.
- An error occurs when a DWARF function is called.

ddpi_machinestate_query_cr_change operation

The `ddpi_machinestate_query_cr_change` operation queries the given `Ddpi_MachineState` object and returns true if any bit setting has been modified on any CR.

Prototype

```
int ddpi_machinestate_query_cr_change(
  Ddpi_MachineState     machinestate,
  Dwarf_Bool*           ret_flag,
  Ddpi_Error*           error);
```

Parameters

machinestate
Type: `Ddpi_MachineState`

Input. This accepts the `Ddpi_MachineState` object.

***ret_flag**
Type: `Dwarf_Bool`

Output. This returns true if the state of the CR has changed.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the flag that indicates whether a change in CR has occurred.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The Ddpi_Space or Ddpi_Info object associated with `machinestate` is NULL.
- An error occurs when a DWARF function is called.

ddpi_machinestate_get_psw operation

The `ddpi_machinestate_get_psw` operation returns specific PSW elements of the given `Ddpi_MachineState` object.

`ddpi_machinestate_get_psw` returns the following elements:

- The value of the requested PSW type.
- The valid flag for the given PSW.

  Note: The PSW as a whole can be invalidated if a invalid AMODE or IP is set.
- The modified flag for the given PSW, which indicates whether the PSW value has been modified since the last reset, or if the AMODE or IP have been modified.
- The stored location of this given PSW, which may be used when shadowing PSW values.

Prototype

```
int ddpi_machinestate_get_psw(
  Ddpi_PSW_Type         psw_type,
  Ddpi_MachineState     machinestate,
  Ddpi_PSW*             ret_psw_value,
  Dwarf_Bool*           ret_psw_valid,
  Dwarf_Bool*           ret_psw_modified,
  Dwarf_Unsigned*       ret_psw_locn,
  Ddpi_Error*           error);
```

Parameters

psw_type
Input. This accepts the PSW type.

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_psw_value
Output. This returns the PSW value.

ret_psw_valid
Output. This returns the PSW valid flag.

ret_psw_modified
Output. This returns the PSW modified flag.

ret_psw_locn
Output. This returns the PSW location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the PSW values.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- `ret_psw_value`, `ret_psw_valid`, `ret_psw_modified`, or `ret_psw_locn` is NULL
- `psw_number` is out of range
- An error occurs when a DWARF function is called.

ddpi_machinestate_set_psw operation

The `ddpi_machinestate_set_psw` operation updates the given PSW of the given `Ddpi_MachineState` object.

`ddpi_machinestate_set_psw` updates the PSW value, flag, and location.

Prototype

```
int ddpi_machinestate_set_psw(
  Ddpi_PSW_Type         psw_type,
  Ddpi_MachineState     machinestate,
  Ddpi_PSW              psw_value,
  Dwarf_Bool            psw_valid,
  Dwarf_Unsigned        psw_locn,
  Ddpi_Error*           error);
```

Parameters

psw_type
Input. This accepts the PSW type.

machinestate
Input. This accepts the `Ddpi_MachineState` object.

psw_value
Input. This accepts the new PSW value.

psw_valid
Input. This accepts the new PSW valid flag.

psw_locn
Input. This accepts the new PSW location, or 0.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the PSW values.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- An unknown value is given for the valid flag
- An error occurs when a DWARF function is called.

ddpi_machinestate_query_psw_change operation

The `ddpi_machinestate_query_psw_change` operation queries the given `Ddpi_MachineState` object and returns true if the a bit setting has been modified on any PSW.

A bit setting is modified whenever the PSW, AMODE or IP was set.

Prototype

```
int ddpi_machinestate_query_psw_change(
  Ddpi_MachineState     machinestate,
  Dwarf_Bool*           ret_flag,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_flag
Output. This returns true if the state of the PSW has changed.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the flag that indicates whether a change in PSW has occurred.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- An error occurs when a DWARF function is called.

ddpi_machinestate_get_ip operation

The ddpi_machinestate_get_ip operation returns specific IP elements of the given Ddpi_MachineState object.

ddpi_machinestate_get_ip returns the following elements:

- The value of the current IP in the Ddpi_MachineState
- The valid flag for the given IP. True is returned if the value of the current IP is known to be valid. The IP can be invalidated by calling ddpi_machinestate_set_ip or ddpi_machinestate_set_psw with an invalid value
- The modified flag for the given IP, which indicates whether the IP value has been modified since creation or the last reset. The value can also be modified by calls to ddpi_machinestate_set_psw for the current IP

Prototype

```
int ddpi_machinestate_get_ip(
  Ddpi_MachineState     machinestate,
  Dwarf_Addr*           ret_ip,
  Dwarf_Bool*           ret_ip_valid,
  Dwarf_Bool*           ret_ip_modified,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the Ddpi_MachineState object.

ret_ip
Output. This returns the IP value.

ret_ip_valid
Output. This returns the IP valid flag.

ret_ip_modified
Output. This returns the IP modified flag.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the IP values.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- machinestate is NULL
- The Ddpi_Space or Ddpi_Info object associated with machinestate is NULL.
- ret_ip_value, ret_ip_valid, or ret_ip_modified is NULL
- An error occurs when a DWARF function is called.

ddpi_machinestate_set_ip operation

The ddpi_machinestate_set_ip operation updates the IP of the given Ddpi_MachineState object.

Note: An invalid IP will invalidate the whole PSW.

Prototype

```
int ddpi_machinestate_set_ip(
  Ddpi_MachineState     machinestate,
  Dwarf_Addr            new_ip,
  Dwarf_Bool            new_ip_valid,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the Ddpi_MachineState object.

new_ip
Input. This accepts the new IP.

new_ip_valid
Input. This accepts the new IP valid flag.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful update of the IP values.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- machinestate is NULL
- The Ddpi_Space or Ddpi_Info object associated with machinestate is NULL.
- An unknown value is given for the valid flag
- An error occurs when a DWARF function is called.

ddpi_machinestate_get_amode operation

The ddpi_machinestate_get_amode operation returns specific AMODE elements of the given Ddpi_MachineState object.

The ddpi_machinestate_get_amode operation returns the following elements:

- The value of the current AMODE.
- The validation flag for the AMODE. For example, if the value of the current AMODE is known to be valid, the operation returns true.

  Note: The AMODE can be invalidated by calling ddpi_machinestate_set_amode or ddpi_machinestate_set_psw with an invalid value.
- The modification flag for the AMODE, which indicates whether the AMODE value has been modified since its creation or most recent reset.

Note: The value can also be modified by calls to `ddpi_machinestate_set_psw` for the current AMODE.

Prototype

```
int ddpi_machinestate_get_amode(
  Ddpi_MachineState     machinestate,
  Ddpi_Addr_Mode*       ret_amode,
  Dwarf_Bool*           ret_amode_valid,
  Dwarf_Bool*           ret_amode_modified,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_amode
Output. This returns the AMODE value.

ret_amode_valid
Output. This returns the AMODE valid flag.

ret_amode_modified
Output. This returns the AMODE modified flag.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the AMODE values.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- `ret_amode`, `ret_amode_valid`, or `ret_amode_modified` is NULL
- An error occurs when a DWARF function is called.

ddpi_machinestate_set_amode operation

The `ddpi_machinestate_set_amode` operation updates the AMODE of the given `Ddpi_MachineState` object.

An invalid AMODE will invalidate the whole PSW.

Prototype

```
int ddpi_machinestate_set_amode(
  Ddpi_MachineState     machinestate,
  Ddpi_Addr_Mode        new_amode,
  Dwarf_Bool            new_amode_valid,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

new_amode
Input. This accepts the new AMODE.

new_amode_valid
Input. This accepts the new AMODE valid flag.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful update of the AMODE values.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- An unknown value is given for the valid flag
- An error occurs when a DWARF function is called.

ddpi_machinestate_get_space operation

The `ddpi_machinestate_get_space` operation returns the current `Ddpi_Space` object in the given `Ddpi_MachineState` object.

Prototype

```
int ddpi_machinestate_get_space(
  Ddpi_MachineState     machinestate,
  Ddpi_Space*           ret_space,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_space
Output. This returns the `Ddpi_Space` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the SPACE values.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- `ret_space` is NULL.
- An error occurs while creating the nested `Ddpi_StorageLocn`object.

ddpi_machinestate_set_space operation

The ddpi_machinestate_set_space operation associates the given Ddpi_Space object with the given Ddpi_MachineState object.

ddpi_machinestate_set_space will also update the Ddpi_Space value associated with the Ddpi_StorageLocn that is nested inside the given Ddpi_MachineState.

Prototype

```
int ddpi_machinestate_set_space(
  Ddpi_MachineState     machinestate,
  Ddpi_Space            space,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the Ddpi_MachineState object.

space
Input. This accepts the Ddpi_Space object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful update of the SPACE values.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- machinestate is NULL
- The Ddpi_Space or Ddpi_Info object associated with machinestate is NULL.
- An error occurs when updating space in the Ddpi_StorageLocn object.
- An error occurs when a DWARF function is called.

ddpi_machinestate_get_storagelocn operation

The ddpi_machinestate_get_storagelocn operation returns the Ddpi_StorageLocn object nested within the Ddpi_MachineState object.

Do not call ddpi_storagelocn_term to delete Ddpi_StorageLocn. This Ddpi_StorageLocn is terminated when the owning Ddpi_MachineState is terminated.

Prototype

```
int ddpi_machinestate_get_storagelocn(
  Ddpi_MachineState     machinestate,
  Ddpi_StorageLocn*     ret_locn,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the Ddpi_MachineState object.

ret_locn
Output. This returns the Ddpi_StorageLocn object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the STORAGELOCN values.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- machinestate is NULL
- The Ddpi_Space or Ddpi_Info object associated with machinestate is NULL.
- ret_locn is NULL
- An error occurs when a DWARF function is called.

ddpi_machinestate_get_context operation

The ddpi_machinestate_get_context operation returns the current context address and type associated with a Ddpi_MachineState object.

Prototype

```
int ddpi_machinestate_get_context(
  Ddpi_MachineState     machinestate,
  Ddpi_Context_Type*    ret_type,
  Ddpi_Context*         ret_context,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the Ddpi_MachineState object.

ret_type
Output. This returns the Ddpi_Context_Type object.

ret_context
Output. This returns the Ddpi_Context value.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the CONTEXT address and type.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- machinestate is NULL
- The Ddpi_Space or Ddpi_Info object associated with machinestate is NULL.
- ret_context or ret_type is NULL.

- An error occurs when a DWARF function is called.

ddpi_machinestate_set_context operation

The ddpi_machinestate_set_context operation assigns the given context and context type to the Ddpi_StorageLocn object associated with the Ddpi_MachineState object.

Prototype

```
int ddpi_machinestate_set_context(
  Ddpi_MachineState     machinestate,
  Ddpi_Context_Type     type,
  Ddpi_Context          context,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the Ddpi_MachineState object.

type
Input. This accepts the new Ddpi_Context_Type object.

context
Input. This accepts the new Ddpi_Context value.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful update of the CONTEXT address and type.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- machinestate is NULL
- The Ddpi_Space or Ddpi_Info object associated with machinestate is NULL.
- ret_context or ret_type is NULL.
- An error occurs when a DWARF function is called.

ddpi_machinestate_reset_change operation

The ddpi_machinestate_reset_change operation resets all modified flags in the Ddpi_MachineState object.

Prototype

```
int ddpi_machinestate_reset_change(
  Ddpi_MachineState     machinestate,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the Ddpi_MachineState object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful update of the modified flag values in the machine state object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- An error occurs when a DWARF function is called.

ddpi_machinestate_any_change operation

The `ddpi_machinestate_any_change` operation queries if there have been any changes to the machine state since the last call to `ddpi_machinestate_reset`.

`ddpi_machinestate_term` releases all internal resources associated with the machine state object, and invalidates the given machine state.

Prototype

```
int ddpi_machinestate_any_change(
  Ddpi_MachineState     machinestate,
  Dwarf_Bool*           ret_bool,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

ret_bool
Output. This returns true if the state of the `Ddpi_MachineState` has changed.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the information indicating whether the flags in the machine state object have changed.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `machinestate` is NULL
- The `Ddpi_Space` or `Ddpi_Info` object associated with `machinestate` is NULL.
- An error occurs when a DWARF function is called.

Chapter 25. Ddpi_PPA_Extract APIs

Ddpi_PPA_Extract APIs to help you manage the different PPA formats and conditions. They extract the same data regardless of the version of PPA that is being used. Use them only with IBM C/C++ compilation units (CUs).

Ddpi_PPA1_data_struct_s object

Contains the PPA1 data structure definition.

Type definition

```
#define PPA1_DATA_STRUCT_VERSION 0x01
typedef struct Ddpi_PPA1_data_struct_s {
   Dwarf_Small        ppa1_data_struct_version;
   Dwarf_Small        ppa1_version;
   Dwarf_Bool         ppa1_has_off_to_len_of_name;
   Dwarf_Small        ppa1_offset_to_length_of_name;
   Dwarf_Bool         ppa1_external_proc;
   Dwarf_Bool         ppa1_primary_ep;
   Dwarf_Bool         ppa1_LE_dsa;
   Dwarf_Bool         ppa1_Library;
   Dwarf_Bool         ppa1_sampling;
   Dwarf_Bool         ppa1_dsa_exit;
   Dwarf_Bool         ppa1_is_64_bit;
   Dwarf_Bool         ppa1_exceptx;
   Dwarf_Bool         ppa1_PPA3_full;
   Dwarf_Small        ppa1_flag2;
   Dwarf_Bool         ppa1_argparse;
   Dwarf_Bool         ppa1_redir;
   Dwarf_Bool         ppa1_execops;
   Dwarf_Small        ppa1_Linkage;
   Dwarf_Off          ppa1_offset_to_PPA2;
   Dwarf_Bool         ppa1_has_addr_to_PPA3;
   Dwarf_Off          ppa1_addr_to_PPA3;
   Dwarf_Half         ppa1_GPR_save_mask;
   Dwarf_Bool         ppa1_state_table;
   Dwarf_Bool         ppa1_dll;
   Dwarf_Bool         ppa1_SF_layout;
   Dwarf_Small        ppa1_calling;
   Dwarf_Bool         ppa1_sleaf;
   Dwarf_Bool         ppa1_old_code;
   Dwarf_Unsigned     ppa1_max_space;
   Dwarf_Small        ppa1_save;
   Dwarf_Bool         ppa1_async_cond;
   Dwarf_Bool         ppa1_word0_SF_init;
   Dwarf_Bool         ppa1_glue_code;
   Dwarf_Bool         ppa1_return;
   Dwarf_Bool         ppa1_argument;
   Dwarf_Bool         ppa1_register12;
   Dwarf_Bool         ppa1_vararg;
   Dwarf_Bool         ppa1_async_interrupts;
   Dwarf_Off          ppa1_offset_code_descriptor;
   Dwarf_Unsigned     ppa1_parm_length;
   Dwarf_Unsigned     ppa1_prolog_length;
   Dwarf_Small        ppa1_alloca_reg;
   Dwarf_Off          ppa1_off_stack_pointer_update;
   Dwarf_Unsigned     ppa1_length_of_code;
   Dwarf_Bool         ppa1_has_offset_state_variable;
   Dwarf_Off          ppa1_offset_state_variable;
   Dwarf_Bool         ppa1_has_arg_area_length;
   Dwarf_Off          ppa1_arg_area_length;
   Dwarf_Bool         ppa1_has_saved_FPR_mask;
   Dwarf_Half         ppa1_saved_FPR_mask;
```

© Copyright IBM Corp. 2004, 2009

```
   Dwarf_Bool         ppa1_has_saved_AR_mask;
   Dwarf_Half         ppa1_saved_AR_mask;
   Dwarf_Bool         ppa1_has_FPR_saved_locator;
   Dwarf_Unsigned     ppa1_FPR_saved_locator;
   Dwarf_Bool         ppa1_has_AR_saved_locator;
   Dwarf_Unsigned     ppa1_AR_saved_locator;
   Dwarf_Bool         ppa1_has_member_word;
   Dwarf_Unsigned     ppa1_member_word;
   Dwarf_Bool         ppa1_has_intf_mapping_flags;
   Dwarf_Unsigned     ppa1_intf_mapping_flags;
   Dwarf_Bool         ppa1_has_java_method_locator;
   Dwarf_Unsigned     ppa1_java_method_locator;
   Dwarf_Unsigned     ppa1_PPA3_offset;
   Dwarf_Addr         ppa1_address;
} Ddpi_PPA1_Data_Struct;
```

Ddpi_PPA2_data_struct_s object

Contains the PPA2 data structure definition.

Type definition

```
#define PPA2_DATA_STRUCT_VERSION 0x01
typedef struct Ddpi_PPA2_data_struct_s {
   Dwarf_Small        ppa2_data_struct_version;
   Dwarf_Small        ppa2_version;
   Dwarf_Small        ppa2_member_id;
   Dwarf_Small        ppa2_sub_member_id;
   Dwarf_Small        ppa2_member_defined;
   Dwarf_Off          ppa2_offset_to_CEESTART;
   Dwarf_Off          ppa2_offset_to_PPA4;
   Dwarf_Off          ppa2_offset_to_timestamp;
   Dwarf_Off          ppa2_offset_to_primary_epa;
   Dwarf_Bool         ppa2_binary_float;
   Dwarf_Bool         ppa2_library_code;
   Dwarf_Bool         ppa2_service_info;
   Dwarf_Bool         ppa2_md5_sig_in_timestamp_area;
   Dwarf_Addr         ppa2_address;
} Ddpi_PPA2_Data_Struct;
```

ddpi_ppa_extract_ppa1 operation

The `ddpi_ppa_extract_ppa1` operation extracts the PPA1 into a preallocated structure with the version token set.

If the version token is not set or is not recognized, the current version of the function will be used.

Prototype

```
int ddpi_ppa_extract_ppa1 (
  Ddpi_Space               space
  Dwarf_Addr               ppa1_addr,
  Dwarf_Small              data_struct_version,
  Ddpi_PPA1_Data_Struct*   ppa1_data_struct,
  Ddpi_Error*              error);
```

Parameters

space
Input. This accepts the address `Ddpi_Space` object.

ppa1_addr
Input. This accepts the PPA1 address.

data_struct_version
Input. This accepts the PPA1_DATA_STRUCT_VERSION.

ppa1_data_struct
Input/Output. This accepts and returns the allocated PPA1 data structure.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the PPA1 address.

DW_DLV_NO_ENTRY
Returned when the PPA1 does not have the language environment signature.

DW_DLV_ERROR
This value is returned if:

- space is NULL.
- The Ddpi_Info object associated with space is NULL.
- ppa1_addr isDwarf_NULL.
- ppa1_data_struct is NULL.
- data_struct_version is unrecognized.
- An error occurs during memory access.

ddpi_ppa_extract_ppa2 operation

The ddpi_ppa_extract_ppa2 operation extracts the PPA2 into a preallocated structure with the version token set.

If the version token is not set or not recognized, the current version of the function will be used.

Prototype

```
int ddpi_ppa_extract_ppa2 (
  Ddpi_Space               space
  Dwarf_Addr               ppa2_addr,
  Dwarf_Small              data_struct_version,
  Ddpi_PPA2_Data_Struct*   ppa2_data_struct,
  Ddpi_Error*              error);
```

Parameters

space
Input. This accepts the address Ddpi_Space object.

ppa2_addr
Input. This accepts the PPA2 address.

data_struct_version
Input. This accepts the PPA2_DATA_STRUCT_VERSION.

ppa2_data_struct
Input/Output. This accepts and returns the allocated PPA2 data structure.

error
See “The libddpi error parameter” on page 13.

Return values

The ddpi_ppa_extract_ppa2 operation returns DW_DLV_OK upon successful return of the PPA2 address. The ddpi_ppa_extract_ppa2 operation returns DW_DLV_ERROR if:

- space is NULL
- Ddpi_Info associated with space is NULL
- ppa2_addr is Dwarf_NULL
- ppa2_data_struct is NULL
- data_struct_version is unrecognized
- An error occurs while accessing memory

Note: ddpi_ppa_extract_ppa2 never returns DW_DLV_NO_ENTRY.

ddpi_ppa_extract_ppa1_entrypoint_name operation

The ddpi_ppa_extract_ppa1_entrypoint_name operation extracts and returns the PPA1 entry-point name.

ddpi_ppa_extract_ppa1_entrypoint_name takes a ppa1_data_struct and returns a buffer containing the function name from the PPA1 as a NULL-terminated string. It is the responsibility of the user to delete the name buffer. The following is an example of the deletion code:

```
rc = ddpi_dealloc(info, *ret_entrypt_name, DDPI_DLA_STRING)
```

Prototype

```
int ddpi_ppa_extract_ppa1_entrypoint_name(
    Ddpi_Space  space,
    Ddpi_PPA1_Data_Struct* ppa1_data_struct,
    char**      ret_entrypt_name,
    Ddpi_Error*  error);
```

Parameters

space
Input. This accepts the address Ddpi_Space object.

ppa1_data_struct
Input. This accepts the PPA1 data structure containing the required extract and PPA1-address data.

ret_entrypt_name
Output. This returns the PPA1 entry-point name.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the PPA2 entry-point name.

DW_DLV_NO_ENTRY
Returned if there is no entry-point name.

DW_DLV_ERROR
This value is returned if:

- space is NULL.

- The Ddpi_Info object associated with space is NULL.
- ppa1_data_struct or ret_entrypt_name is NULL.
- ppa1_data_struct is NULL.
- data_struct_version is unrecognized.
- An error occurs during memory access.
- An error occurs during memory allocation.

ddpi_ppa_extract_CU_primary_source operation

The ddpi_ppa_extract_CU_primary_source operation extracts the first entry-point name from the PPA4, if the name is present.

ddpi_ppa_extract_CU_primary_source takes a ppa2_data_struct and returns a buffer containing the primary-source file name as a NULL-terminated string.

You must delete the name buffer after the value is returned finished.

The following is an example of the code to delete a name buffer:

```
rc = ddpi_dealloc(info, *ret_source_name, DDPI_DLA_STRING)
```

Prototype

```
ddpi_ppa_extract_CU_primary_source(
    Ddpi_Space     space,
    Ddpi_PPA2_Data_Struct*
                   ppa2_data,
    char**         ret_source_name,
    Ddpi_Error*    error);
```

Parameters

space
Input. This accepts the address Ddpi_Space object.

ppa2_data
Input. This accepts a PPA2 data structure containing the required extract data.

ret_source_name
Output. This returns the entry-point name.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the PPA4 source data.

DW_DLV_NO_ENTRY
Returned if:

- The CU source name is unavailable
- The given CU was compiled with the DEBUG(FORMAT(CDA)) compiler option

DW_DLV_ERROR
This value is returned if:

- space is NULL.
- The Ddpi_Info object associated with space is NULL.

- ppa2_data_struct or ret_source_name is NULL.
- ppa1_data_struct is NULL.
- data_struct_version is unrecognized.
- An error occurs during memory access.
- An error occurs during memory allocation.

ddpi_ppa2_md5_sig operation

The ddpi_ppa2_md5_sig operation either extracts the MD5 signature (if the CU was compiled to create CDA debugging information) or creates an MD5 signature to be used with the converter.

ddpi_ppa2_md5_sig uses a given PPA2 address and PPA1 address list to:

- Extract the compiler-generated source from the given module image, for CDA debugging information
- Create an MD5 signature to match the converted source, for ISD debug information

The given ppa1_list is returned in descending address order.

Prototype

```
int ddpi_ppa2_md5_sig(
  Ddpi_Space            space,
  Dwarf_Addr *          ppa1_list,
  int                   ppa1_elements,
  Dwarf_Addr            ppa2_addr,
  unsigned char         digest[16],
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the address Ddpi_Space object.

ppa1_list
Input. This accepts the list of PPA1 addresses for the CU.

ppa1_elements
Input. This accepts the number of PPA1 elements for the CU.

ppa2_addr
Input. This accepts the PPA2 addresses for the CU.

digest[16]
Output. This returns the PPA2 MD5 signature.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval or creation of the MD5 signature.

DW_DLV_NO_ENTRY
Returned if:

- PPA1 does not have a pointer to name.
- PPA1 does not have the language environment signature.

DW_DLV_ERROR

This value is returned if:

- `space` is NULL.
- The `Ddpi_Info` object associated with `space` is NULL.
- `ppa2_addr` is NULL.
- `ppa1_list` is NULL and `ppa1_elements` are greater than 1.
- An error occurs during access of the given PPA1s or PPA2s.
- An error occurs when an allocation of temporary space failed.

Chapter 26. Ddpi_StackState APIs

Every function of an application is allocated space in the stack. This space is referred to as a stack frame, or dynamic storage area (DSA). `ddpi_stackstate` operations create, format, and terminate objects in order to read and write stack-frame information. `Ddpi_StackState` objects store different elements of the stack frame.

The CDA library provides three types of APIs to manage stack frame information:

Ddpi_StackState APIs
The `ddpi_stackstate` operations:

- Create `Ddpi_StackState` objects to contain different elements of the stack frame.
- Use the `Ddpi_Stackstate_Fn` object when they call a program analysis application (user-supplied) stack-state handler function to load the stack-state information.
- Use the program analysis application stack-state handler functions to load the information. Stack-state handler functions identify the stack type and extract stack-state information.

Ddpi_Stackstate_Fn APIs
The `ddpi_stackstate_fn` operations:

- Create the `Ddpi_Stackstate_Fn` object.
- Register the stack-state handler functions that are available in the program analysis application by adding them to the `Ddpi_Stackstate_Fn` object.

Standard stack state-handler APIs for Language Environment
Stack-state handlers collect stack-frame information as they walk the stack, and load it into the `Ddpi_StackState` objects. The CDA library provides two operations that read the stack frames that conform to Language Environment. You can use these operations as prototypes to create other stack-state handling functions.

Note: For more information about z/OS stack frames, see *z/OS Language Environment Vendor Interfaces, SA22-7568*.

Ddpi_StackState object

A `Ddpi_StackState` object is an opaque data type that contains information about a single stack frame.

Each `Ddpi_StackState` object tracks the following information:

- The stack frame format
- The type of stack frame convention
- The types of stack linkage
- Location of the DSA
- DSA length
- Allocate base
- Location of the parent
- Location of the `alloca` area
- Location of the PPA1 block

© Copyright IBM Corp. 2004, 2009

- Location of the PPA2 block

An instance of the Ddpi_StackState type is created as a result of a successful call to the ddpi_stackstate_create operation. The initial state of a Ddpi_StackState object can be any of the following:

- The stack format, type, and linkage are unknown
- The DSA location is 0, the DSA length is 0
- The parent stack, initial alloca base, PPA1 block, and PPA2 block location are all 0

The storage pointed to by this descriptor will be freed during the execution of the ddpi_stackstate_term operation. An instance of the Ddpi_StackState object can be reset to the initial state by the ddpi_stackstate_init operation.

Type definition

```
typedef struct Ddpi_StackState_s*    Ddpi_StackState;
```

Ddpi_Stack_Format object

This contains the stack frame format, which indicates the direction in which the stack will grow.

Type definition

```
typedef enum Ddpi_Stack_Format_s {
  Ddpi_STSF_Unknown  =  0,
  Ddpi_STSF_UP       =  1,
  Ddpi_STSF_DOWN     =  2,
  Ddpi_STSF_NONE     =  3
} Ddpi_Stack_Format;
```

Members

Ddpi_STSF_Unknown
If this value is 0, it will not be accepted by the ddpi_stackstate_create operation.

Ddpi_STSF_UP
If this value is 1, the DSA format suits a NOXPLINK-compiled program that uses an upward growing stack.

Ddpi_STSF_DOWN
If this value is 2, the DSA format suits ax XPLINK-compiled program that uses an downward growing stack.

Ddpi_STSF_NONE
If this value is 3, the DSA format does not suit the program.

Ddpi_Stack_Type object

This contains the type of stack-frame convention.

Type definition

```
typedef enum Ddpi_Stack_Type_s {
  Ddpi_STST_Unknown       = 0,
  Ddpi_STST_31_NOXPLINK   = 1,
  Ddpi_STST_31_XPLINK     = 2,
```

```
  Ddpi_STST_64_XPLINK     = 3,
  Ddpi_STST_31_TYPE1      = 4,
  Ddpi_STST_64_TYPE1      = 5
} Ddpi_Stack_Type;
```

Members

Ddpi_STST_Unknown
If this value is 0, it will not be accepted by ddpi_stackstate_create.

Ddpi_STST_31_NOXPLINK
If this value is 1, the stack type is 31-bit NOXPLINK.

Ddpi_STST_31_XPLINK
If this value is 2, the stack type is 31-bit XPLINK.

Ddpi_STST_64_XPLINK
If this value is 3, the stack type is 64-bit XPLINK.

Ddpi_STST_31_TYPE1
If this value is 4, the stack type is 31-bit Type-1.

Ddpi_STST_64_TYPE1
If this value is 5, the stack type is 64-bit Type-1.

Ddpi_Stack_Linkage object

This contains the types of stack linkage.

Type definition

```
typedef enum Ddpi_Stack_Linkage_s {
  Ddpi_STSL_Unknown       =  0,
  Ddpi_STSL_PRIVATE       =  1,
  Ddpi_STSL_FSTLNK        =  2,
  Ddpi_STSL_XPLINK        =  3,
  Ddpi_STSL_OS            =  4,
  Ddpi_STSL_PLI           =  5,
  Ddpi_STSL_TYPE1         =  6
} Ddpi_Stack_Linkage;
```

Members

Ddpi_STSL_Unknown
This value will not be accepted by the `ddpi_stackstate_create` operation.

Ddpi_STSL_PRIVATE
If this value is 1, the stack linkage is C (NOXPLINK, 31-bit).

Ddpi_STSL_FSTLNK
If this value is 2, the stack linkage is C++ (NOXPLINK, 31-bit).

Ddpi_STSL_XPLINK
If this value is 3, the stack linkage is XPLINK (31-bit, 64-bit).

Ddpi_STSL_OS
If this value is 4, the stack linkage is OS.

Ddpi_STSL_PLI
If this value is 5, the stack linkage is PL/I.

Ddpi_STSL_TYPE1
If this value is 6, the stack linkage is Type-1 (31-bit, 64-bit).

Ddpi_StackState_Identify_Handler object

The Ddpi_StackState_Identify_Handler object defines the prototype for a stack-state handler function that gathers information from the initial stack frame.

The program analysis application should provide different versions of stack-state handler functions. As a convenience, CDA supplies the ddpi_stackstate_identify_le operation to identify the stack state for Language Environment applications.

Type definition

```
typedef int (*Ddpi_StackState_Identify_Handler) (
   Ddpi_MachineState     machinestate,
   Ddpi_StackState       stackstate,
   Dwarf_Ptr             workarea,
   unsigned int          workarea_len,
   Ddpi_Error*           error);
```

Members

machinestate
Input. This accepts the Ddpi_MachineState object.

stackstate
Input. This accepts the Ddpi_StackState object.

workarea
Input. This accepts a pointer to the work area buffer.

workarea_len
Input. This accepts the work area length.

error
See “The libddpi error parameter” on page 13.

Ddpi_StackState_Parent_Handler object

The Ddpi_StackState_Parent_Handler object defines the prototype for a stack-state handler function that gathers information from the parent stack frame. Several versions of the function should be based on this prototype.

As a convenience, CDA provides the “ddpi_stackstate_identify_le operation” on page 271 operation to gather information from the parent stack frame when Language Environment linkages are in effect.

Type definition

```
typedef int (*Ddpi_StackState_Parent_Handler) (
   Ddpi_MachineState     machinestate,
   Ddpi_StackState       stackstate,
   Ddpi_MachineState     parent_machinestate,
   Ddpi_StackState       parent_stackstate,
   Dwarf_Ptr             workarea,
   unsigned int          workarea_len,
   Ddpi_Error*           error);
```

Members

machinestate
Input. This accepts the child Ddpi_MachineState object.

stackstate
Input. This accepts the child `Ddpi_StackState` object.

parent_machinestate
Output. This returns the parent `Ddpi_MachineState` object.

parent_stackstate
Output. This returns the parent `Ddpi_StackState` object.

workarea
Input. This accepts a pointer to the work-area buffer.

workarea_len
Input. This accepts the work-area length.

error
See “The libddpi error parameter” on page 13.

ddpi_stackstate_create operation

The `ddpi_stackstate_create` operation allocates sufficient contiguous memory for a `Ddpi_StackState` object, and initializes that object.

Prototype

```
int ddpi_stackstate_create(
  Ddpi_Info             info,
  Ddpi_StackState*      ret_stackstate,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

ret_stackstate
Output. This returns the `Ddpi_StackState`.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the stack-state object has been successfully initialized.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `info` is NULL
- `ret_stackstate` is NULL
- An error occurs during memory allocation.

ddpi_stackstate_term operation

The `ddpi_stackstate_term` operation releases all the memory associated by the `Ddpi_StackState` object.

Prototype

```
int ddpi_stackstate_term(
  Ddpi_StackState       stackstate,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the resources associated with the stack state object have been successfully released.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `info` is NULL.
- `ret_stackstate` is NULL.
- An error occurs during memory deallocation.

ddpi_stackstate_init operation

The `ddpi_stackstate_init` operation initializes the state of the a `Ddpi_StackState` object.

Prototype

```
int ddpi_stackstate_init(
  Ddpi_StackState       stackstate,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the given state-stack object has been successfully initialized.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL
- An error occurs during memory deallocation.

ddpi_stackstate_identify operation

The ddpi_stackstate_identify operation attempts to identify and dissect the initial stack frame.

The initial stack frame is associated with the target machine state contained in the given Ddpi_MachineState object. ddpi_stackstate_identify uses the operations associated with the Ddpi_MachineState object to examine the register and storage contents.

The outer ddpi_stackstate_identify operation will iterate through the Ddpi_StackState_Fnobject, which is provided by the application. When a stack frame is successfully identified, the Ddpi_StackState object is initialized, and set up with all relevant stack frame-information.

Prototype

```
int ddpi_stackstate_identify(
  Ddpi_StackState_Fn    stackstate_fn,
  Ddpi_MachineState     machinestate,
  Ddpi_StackState       stackstate,
  Dwarf_Ptr             workarea,
  unsigned int          workarea_len,
  Ddpi_Error*           error);
```

Parameters

stackstate_fn
Input. This accepts the Ddpi_StackState_Fn object.

machinestate
Input. This accepts the Ddpi_MachineState object.

stackstate
Input. This accepts the Ddpi_StackState object.

workarea
Input. This accepts the pointer to the work area buffer.

workarea_len
Input. This accepts the work area length.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the given state-stack object has been successfully identified and analyzed.

DW_DLV_NO_ENTRY
Returned if the stack frame is not in a format that is recognized by this exit.

DW_DLV_ERROR
This value is returned if:

- stackstate is NULL.
- The Ddpi_Info object associated with stackstate is NULL
- stackstate_fn is NULL
- machinestate is NULL

ddpi_stackstate_parent operation

The ddpi_stackstate_parent operation attempts to identify and dissect the parent stack frame associated with the target machine state.

This process requires the requires the target machine state and the current stack frame. The target machine state is specified by the given Ddpi_MachineState object. The current stack frame is specified by the given Ddpi_StackState object.

ddpi_stackstate_parent uses the APIs associated with the Ddpi_MachineState object to examine the register and storage contents.

The outer ddpi_stackstate_parent operation will iterate through the Ddpi_StackState_Fn object, which is provided by the application. When a stack frame is successfully identified, the parent Ddpi_MachineState object is initialized and all relevant machine state values are set. If any value is unknown or questionable, the valid flag for that value must be FALSE.

Note: The parent Ddpi_MachineState object may also be cloned from the current Ddpi_MachineState object, as appropriate.

The parent Ddpi_StackState object is then initialized, and set up with all relevant parent stack frame information.

Prototype

```
int ddpi_stackstate_parent(
  Ddpi_StackState_Fn    stackstate_fn,
  Ddpi_MachineState     machinestate,
  Ddpi_StackState       stackstate,
  Ddpi_MachineState     parent_machinestate,
  Ddpi_StackState       parent_stackstate,
  Dwarf_Ptr             workarea,
  unsigned int          workarea_len,
  Ddpi_Error*           error);
```

Parameters

Ddpi_StackState_Fn
Input. This accepts the Ddpi_StackState_Fn object.

machinestate
Input. This accepts the Ddpi_MachineState object.

stackstate
Input. This accepts the Ddpi_StackState object.

parent_machinestate
Output. This returns the parent Ddpi_MachineState object.

parent_stackstate
Output. This returns the parent Ddpi_StackState object.

workarea
Input. This accepts a pointer to the work area buffer.

workarea_len
Input. This accepts the work area length.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when a pointer to the parent of the given state-stack object has been successfully retrieved.

DW_DLV_NO_ENTRY
Returned if the parent stack frame is not in a format that is recognized by this exit.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- `parent_stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.
- `stackstate_fn` is NULL.
- `machinestate` is NULL.
- `parent_machinestate` is NULL.

ddpi_stackstate_get_stack_format operation

The `ddpi_stackstate_get_stack_format` operation returns the `Ddpi_StackState` object format.

Prototype

```
int ddpi_stackstate_get_stack_format(
  Ddpi_StackState       stackstate,
  Ddpi_Stack_Format*    ret_format,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

ret_format
Output. This returns the stack format.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when a pointer to the format of the given state-stack object has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The Ddpi_Info object associated with `stackstate` is NULL.
- `ret_format` is NULL.

ddpi_stackstate_set_stack_format operation

The ddpi_stackstate_set_stack_format operation assigns the given format to the Ddpi_StackState format.

Prototype

```
int ddpi_stackstate_set_stack_format(
  Ddpi_StackState       stackstate,
  Ddpi_Stack_Format     new_format,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the Ddpi_StackState object.

new_format
Input. This accepts the new stack format.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when a pointer to the format of the given state-stack object has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- stackstate is NULL.
- The Ddpi_Info object associated with stackstate is NULL.
- ret_format is NULL.

ddpi_stackstate_get_frame_type operation

The ddpi_stackstate_get_frame_type operation returns the Ddpi_StackState type.

Prototype

```
int ddpi_stackstate_get_frame_type(
  Ddpi_StackState       stackstate,
  Ddpi_Stack_Type*      ret_type,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the Ddpi_StackState object.

ret_type
Output. This returns the stack frame type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when a pointer to the frame of the given state-stack object has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.
- `ret_type` is NULL.

ddpi_stackstate_set_frame_type operation

The `ddpi_stackstate_set_frame_type` operation assigns the given type to the `Ddpi_StackState` object type.

Prototype

```
int ddpi_stackstate_set_frame_type(
  Ddpi_StackState       stackstate,
  Ddpi_Stack_Type       new_type,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

new_type
Input. This accepts the new stack frame type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when a pointer to the frame of the given state-stack object has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.

ddpi_stackstate_get_linkage operation

The `ddpi_stackstate_get_linkage` operation returns the `Ddpi_StackState` object linkage.

Prototype

```
int ddpi_stackstate_get_linkage(
  Ddpi_StackState       stackstate,
  Ddpi_Stack_Linkage*   ret_linkage,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

ret_linkage
Output. This returns the subprogram linkage.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the object linkage of the given state-stack object has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.
- `ret_linkage` is NULL.

ddpi_stackstate_set_linkage operation

The `ddpi_stackstate_set_linkage` operation assigns the given subprogram linkage to the `Ddpi_StackState` linkage.

Prototype

```
int ddpi_stackstate_set_linkage(
  Ddpi_StackState       stackstate,
  Ddpi_Stack_Linkage    new_linkage,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

new_linkage
Input. This accepts the new subprogram linkage.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the object linkage of the given state-stack object has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.

ddpi_stackstate_get_dsa_locn operation

The `ddpi_stackstate_get_dsa_locn` operation returns the `Ddpi_StackState` DSA location.

Prototype

```
int ddpi_stackstate_get_dsa_locn(
  Ddpi_StackState       stackstate,
  Dwarf_Addr*           ret_dsa_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

ret_dsa_locn
Output. This returns the DSA location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when a pointer to the DSA location of the given state-stack object has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.
- `ret_dsa_locn` is NULL.

ddpi_stackstate_set_dsa_locn operation

The `ddpi_stackstate_set_dsa_locn` operation assigns the given location to the `Ddpi_StackState` DSA location.

Prototype

```
int ddpi_stackstate_set_dsa_locn(
  Ddpi_StackState       stackstate,
  Dwarf_Addr            new_dsa_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

new_dsa_locn
Input. This accepts the new DSA location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when a pointer to the DSA location of the given state-stack object has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.

ddpi_stackstate_get_dsa_len operation

The `ddpi_stackstate_get_dsa_len` operation returns the `Ddpi_StackState` DSA length.

Prototype

```
int ddpi_stackstate_get_dsa_len(
  Ddpi_StackState       stackstate,
  Dwarf_Unsigned*       ret_dsa_len,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

ret_dsa_len
Output. This returns the DSA length.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the length of the DSA location of the given state-stack object has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.
- `ret_dsa_len` is NULL.

ddpi_stackstate_set_dsa_len operation

The `ddpi_stackstate_set_dsa_len` operation assigns the given length to the `Ddpi_StackState` DSA length.

Prototype

```
int ddpi_stackstate_set_dsa_len(
  Ddpi_StackState       stackstate,
  Dwarf_Unsigned        new_dsa_len,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

new_dsa_len
Input. This accepts the new DSA length.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the length of the DSA location of the given state-stack object has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.

ddpi_stackstate_get_laa_locn operation

The `ddpi_stackstate_get_laa_locn` operation queries the Library Anchor Area (LAA) address for the current stack frame.

Prototype

```
int ddpi_stackstate_get_laa_locn(
  Ddpi_StackState       stackstate,
  Dwarf_Unsigned*       ret_laa_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

ret_laa_locn
Output. This returns the LAA address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the Library Anchor Area (LAA) address for the current stack frame has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.

ddpi_stackstate_set_laa_locn operation

The `ddpi_stackstate_set_laa_locn` operation assigns the given value as the Library Anchor Area (LAA) address for the current stack frame.

Prototype

```
int ddpi_stackstate_set_laa_locn (
  Ddpi_StackState       stackstate,
  Dwarf_Addr            new_laa_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

new_laa_locn
Input. This accepts the new DSA LAA address.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the Library Anchor Area (LAA) address for the current stack frame has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.

ddpi_stackstate_get_parent_locn operation

The `ddpi_stackstate_get_parent_locn` operation returns the `Ddpi_StackState` parent stack location.

Prototype

```
int ddpi_stackstate_get_parent_locn(
  Ddpi_StackState       stackstate,
  Dwarf_Addr*           ret_parent_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

ret_parent_locn
Output. This returns the parent stack location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the parent stack address for the current stack frame has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.
- `ret_parent_locn` is NULL.

ddpi_stackstate_set_parent_locn operation

The `ddpi_stackstate_set_parent_locn` operation assigns the given parent stack frame location to the `Ddpi_StackState` parent stack location.

Prototype

```
int ddpi_stackstate_set_parent_locn(
  Ddpi_StackState       stackstate,
  Dwarf_Addr            new_parent_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

new_parent_locn
Input. This accepts the new parent stack frame location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the parent stack address for the current stack frame has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.

ddpi_stackstate_get_alloca_base operation

The ddpi_stackstate_get_alloca_base operation returns the Ddpi_StackState object initial alloca_base.

Prototype

```
int ddpi_stackstate_get_alloca_base(
  Ddpi_StackState       stackstate,
  Dwarf_Addr*           ret_alloca_base,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the Ddpi_StackState object.

ret_alloca_base
Output. This returns the alloca_base.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the initial alloca_base for the current stack frame has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- stackstate is NULL.
- The Ddpi_Info object associated with stackstate is NULL.
- ret_alloca_base is NULL.

ddpi_stackstate_set_alloca_base operation

The ddpi_stackstate_set_alloca_base operation assigns the given alloca_base to the Ddpi_StackState initial alloca_base.

Prototype

```
int ddpi_stackstate_set_alloca_base(
  Ddpi_StackState       stackstate,
  Dwarf_Addr            new_alloca_base,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the Ddpi_StackState object.

new_alloca_base
Input. This accepts the new alloca_base.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the initial `alloca_base` for the current stack frame has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.

ddpi_stackstate_get_ep_locn operation

The `ddpi_stackstate_get_ep_locn` operation returns the address of the entry point of the subprogram whose invocation resulted in the creation of the current stack frame.

Prototype

```
int ddpi_stackstate_get_ep_locn(
  Ddpi_StackState       stackstate,
  Dwarf_Addr*         ret_ep_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

ret_ep_locn
Input. This accepts the `address` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the address of the entry point of the subprogram whose invocation resulted in the creation of the current stack frame has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.
- `ret_ep_locn` is NULL.

ddpi_stackstate_set_ep_locn operation

The `ddpi_stackstate_set_ep_locn` operation assigns the given PPA1 location to the entry-point address for subprogram invoked for the current stack frame.

Prototype

```
int ddpi_stackstate_set_ep_locn (
  Ddpi_StackState       stackstate,
  Dwarf_Addr          new_ep_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

new_ep_locn
Input. This accepts the new PPA1 location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the address of the entry point of the subprogram whose invocation resulted in the creation of the current stack frame has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.

ddpi_stackstate_get_ppa1_locn operation

The `ddpi_stackstate_get_ppa1_locn` operation returns the `Ddpi_StackState` PPA1 location.

Prototype

```
int ddpi_stackstate_get_ppa1_locn (
  Ddpi_StackState       stackstate,
  Dwarf_Addr*           ret_ppa1_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

ret_ppa1_locn
Output. This returns the PPA1 location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the address of the PPA1 location has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.
- `ret_ppa1_locn` is NULL.

ddpi_stackstate_set_ppa1_locn operation

The `ddpi_stackstate_set_ppa1_locn` operation assigns the given PPA1 location to the `Ddpi_StackState` PPA1 location.

Prototype

```
int ddpi_stackstate_set_ppa1_locn(
  Ddpi_StackState       stackstate,
  Dwarf_Addr            new_ppa1_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

new_ppa1_locn
Input. This accepts the new PPA1 location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the address of the PPA1 location has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.

ddpi_stackstate_get_ppa2_locn operation

The `ddpi_stackstate_get_ppa2_locn` operation returns the `Ddpi_StackState` PPA2 location.

Prototype

```
int ddpi_stackstate_get_ppa2_locn(
  Ddpi_StackState       stackstate,
  Dwarf_Addr*           ret_ppa2_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

ret_ppa2_locn
Output. This returns the PPA2 location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the address of the PPA2 location has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.
- `ret_ppa2_locn` is NULL.

ddpi_stackstate_set_ppa2_locn operation

The `ddpi_stackstate_set_ppa2_locn` operation assigns the given PPA2 location to the `Ddpi_StackState` PPA2 location.

Prototype

```
int ddpi_stackstate_set_ppa2_locn(
  Ddpi_StackState       stackstate,
  Dwarf_Addr            new_ppa2_locn,
  Ddpi_Error*           error);
```

Parameters

stackstate
Input. This accepts the `Ddpi_StackState` object.

new_ppa2_locn
Input. This accepts the new PPA2 location.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the address of the PPA2 location has been successfully assigned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `stackstate` is NULL.
- The `Ddpi_Info` object associated with `stackstate` is NULL.
- `ret_ppa2_locn` is NULL.

Chapter 27. Ddpi_StackState_Fn APIs

Ddpi_StackState_Fn operations create objects in which to register the available stack-state handler functions that are provided by the program analysis application. These stack-state handler functions are used to identify the stack type and to collect stack-state information. Ddpi_StackState objects store this information. These objects are used whenever a ddpi_stackstate operation calls a user-supplied stack-state handler function.

For a description of all APIs that handle stack-frame information, see Chapter 26, “Ddpi_StackState APIs,” on page 243.

Ddpi_StackState_Fn object

This object is an opaque data type that associates all the identify or parent functions for different types.

Type definition

```
typedef struct Ddpi_StackState_Fn_s* Ddpi_StackState_Fn;
```

ddpi_stackstate_fn_create operation

The ddpi_stackstate_fn_create operation allocates a piece of memory big enough to hold a Ddpi_StackState_Fn object.

Prototype

```
int ddpi_stackstate_fn_create(
  Ddpi_Info             info,
  Ddpi_StackState_Fn*   ret_stackstate_fn,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the Ddpi_Info object.

ret_stackstate_fn
Output. This returns the Ddpi_StackState_Fn object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the Ddpi_StackState_Fn object has been successfully retrieved.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- info is NULL.
- ret_stackstate_fn is NULL.
- An error occurs during memory allocation.

© Copyright IBM Corp. 2004, 2009

ddpi_stackstate_fn_term operation

The ddpi_stackstate_fn_term operation releases all the resources associated with the Ddpi_StackState_Fn object.

Prototype

```
int ddpi_stackstate_fn_term(
  Ddpi_StackState_Fn    stackstate_fn,
  Ddpi_Error*           error);
```

Parameters

stackstate_fn
Input. This accepts the Ddpi_StackState_Fn object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the resources associated with the Ddpi_StackState_Fn object have been successfully released.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- stackstate_fn is NULL.
- The Ddpi_Info object associated with stackstate_fn is NULL.
- An error occurs during termination of child descriptors.
- An error occurs during memory allocation.

ddpi_stackstate_fn_add operation

The ddpi_stackstate_fn_add operation adds a pair of identify and parent handlers to support stack frame queries.

Prototype

```
int ddpi_stackstate_fn_add(
  Ddpi_StackState_Fn    stackstate_fn,
  Ddpi_StackState_Identify_Handler
                        identify_handler,
  Ddpi_StackState_Parent_Handler
                        parent_handler,
  Ddpi_Error*           error);
```

Parameters

stackstate_fn
Input. This accepts the Ddpi_StackState_Fn object.

identify_handler
Input. This accepts the stack-state handler function that will gather the information for the initial stack-frame. For example, if the stack frame conforms to the LE, then you can use ddpi_stackstate_identify_le.

parent_handler
Input. This accepts the stack-state handler function that will gather the

information for the parent stack-frame. For example, if the stack frame conforms to the LE, then you can use ddpi_stackstate_parent_le.

error
See “The libddpi error parameter” on page 13.

Return values

The ddpi_stackstate_fn_add operation returns DW_DLV_OK when the handlers to support stackframe queries have been successfully added. The ddpi_stackstate_fn_add operation returns DW_DLV_ERROR if:

- stackstate_fn is NULL
- Ddpi_Info associated with stackstate_fn is NULL
- identify_handler is NULL
- parent_handler is NULL
- An error occurs while deallocating memory

Note: ddpi_stackstate_fn_add never returns DW_DLV_NO_ENTRY.

ddpi_stackstate_fn_get_count operation

The ddpi_stackstate_fn_get_count operation returns the number of elements in the handler list associated with the Ddpi_StackState_Fn object.

Prototype

```
int ddpi_stackstate_fn_get_count(
  Ddpi_StackState_Fn    stackstate_fn,
  int*                  ret_count,
  Ddpi_Error*           error);
```

Parameters

stackstate_fn
Input. This accepts the Ddpi_StackState_Fn object.

ret_count
Output. This returns the number of handlers.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the number of elements in the handler list associated with the Ddpi_StackState_Fn object has been successfully returned.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- stackstate_fn is NULL
- The Ddpi_Info object associated with stackstate_fn is NULL.
- ret_count is NULL.

ddpi_stackstate_fn_get_identify operation

The ddpi_stackstate_fn_get_identify operation returns the address of the specified ddpi_stackstate_identify operation.

Prototype

```
int ddpi_stackstate_fn_get_identify(
  Ddpi_StackState_Fn    stackstate_fn,
  int                   index,
  Ddpi_StackState_Identify_Handler*      /* -
                        ret_identify_handler,
  Ddpi_Error*           error);
```

Parameters

stackstate_fn
Input. This accepts the Ddpi_StackState_Fn object.

index
Input. This accepts the index of the identify function.

ret_identify_handler
Output. This returns the handler of the identify function.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- stackstate_fn is NULL.
- The Ddpi_Info object associated with stackstate_fn is NULL.
- ret_identify_handler is NULL.

ddpi_stackstate_fn_set_identify operation

The ddpi_stackstate_fn_set_identify operation assigns the address of the specified ddpi_stackstate_identify operation to the given index.

Prototype

```
int ddpi_stackstate_fn_set_identify(
  Ddpi_StackState_Fn    stackstate_fn,
  int                   index,
  Ddpi_StackState_Identify_Handler
                        identify_handler,
  Ddpi_Error*           error);
```

Parameters

stackstate_fn
Input. This accepts the Ddpi_StackState_Fn object.

index
Input. This accepts the index of the identify function.

identify_handler
Input. This accepts the new handler of the identify function.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the address of the specified `ddpi_stackstate_identify` operation to the given index has been successfully retrieved.

DW_DLV_NO_ENTRY
Returned if the index is out of range.

DW_DLV_ERROR
This value is returned if:

- `stackstate_fn` is NULL.
- The Ddpi_Info object associated with `stackstate_fn` is NULL.

ddpi_stackstate_fn_get_parent operation

The `ddpi_stackstate_fn_get_parent` operation returns the address of the specified `ddpi_stackstate_parent` operation.

Prototype

```
int ddpi_stackstate_fn_get_parent(
  Ddpi_StackState_Fn    stackstate_fn,
  int                   index,
  Ddpi_StackState_Parent_Handler*
                        ret_parent_handler,
  Ddpi_Error*           error);
```

Parameters

stackstate_fn
Input. This accepts the `Ddpi_StackState_Fn` object.

index
Input. This accepts the index of the parent function.

ret_parent_handler
Output. This returns the parent handler.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the address of the specified `ddpi_stackstate_parent` operation has been successfully retrieved.

DW_DLV_NO_ENTRY
Returned if the index is out of range.

DW_DLV_ERROR
This value is returned if:

- `stackstate_fn` is NULL.
- The `Ddpi_Info` object associated with `stackstate_fn` is NULL.
- `ret_parent_handler` is NULL.

ddpi_stackstate_fn_set_parent operation

The ddpi_stackstate_fn_set_parent operation assigns the address of the specified ddpi_stackstate_parent operation to the given index.

Prototype

```
int ddpi_stackstate_fn_set_parent(
  Ddpi_StackState_Fn    stackstate_fn,
  int                   index,
  Ddpi_StackState_Parent_Handler
                        parent_handler,
  Ddpi_Error*           error);
```

Parameters

stackstate_fn
Input. This accepts the Ddpi_StackState_Fn object.

index
Input. This accepts the index of the parent function.

parent_handler
Input. This accepts the new parent handler.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the address of the specified ddpi_stackstate_parent operation has been successfully retrieved.

DW_DLV_NO_ENTRY
Returned if the index is out of range.

DW_DLV_ERROR
This value is returned if:

- stackstate_fn is NULL
- The Ddpi_Info object associated with stackstate_fn is NULL.

Chapter 28. Operations for Language Environment linkages

Stack-state handler functions walk the stack, gathering the required stack-frame information, and add it into the `Ddpi_StackState` objects. As a convenience, stack-state handler operations are provided for Language Environment linkages. They can also be used as prototypes on which to base alternative versions, depending upon your needs.

For a description of all APIs that handle stack-frame information, see Chapter 26, “Ddpi_StackState APIs,” on page 243.

ddpi_stackstate_identify_le operation

The `ddpi_stackstate_identify_le` operation is a standard handler to identify the initial Language Environment stack frame.

Prototype

```
int ddpi_stackstate_identify_le(
  Ddpi_MachineState     machinestate,
  Ddpi_StackState       stackstate,
  Dwarf_Ptr             workarea,
  unsigned int          workarea_len,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the `Ddpi_MachineState` object.

stackstate
Input. This accepts the `Ddpi_StackState` object.

workarea
Input. This accepts a pointer to the work area buffer.

workarea_len
Input. This accepts the work area length.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if there is an error.

ddpi_stackstate_parent_le operation

The `ddpi_stackstate_parent_le` operation is a standard handler to identify the parent language environment stack frame.

© Copyright IBM Corp. 2004, 2009

Prototype

```
int ddpi_stackstate_parent_le(
  Ddpi_MachineState     machinestate,
  Ddpi_StackState       stackstate,
  Ddpi_MachineState     parent_machinestate,
  Ddpi_StackState       parent_stackstate,
  Dwarf_Ptr             workarea,
  unsigned int          workarea_len,
  Ddpi_Error*           error);
```

Parameters

machinestate
Input. This accepts the child `Ddpi_MachineState` object.

stackstate
Input. This accepts the child `Ddpi_StackState` object.

parent_machinestate
Output. This returns the parent `Ddpi_MachineState` object.

parent_stackstate
Output. This returns the parent `Ddpi_StackState` object.

workarea
Input. This accepts a pointer to the work area buffer.

workarea_len
Input. This accepts the work area length.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful completion of the operation.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if there is an error.

Chapter 29. Ddpi_Storage APIs

The Ddpi_Storage APIs provide the only way for the program analysis application to access user memory.

This interface provides the following benefits:

- Consistent access to local, remote, and recorded storage locations without requiring significant modifications to the program analysis application.
- Protection when an operation involving the storage is not successful.
- The ability to filter storage access.

There are two types of Ddpi_Storage objects.

- Ddpi_StorageLocn points to data in memory.
- Ddpi_SavedStorage represents a block of address space.

Both objects use the address space, but there is no other relationship between them. That is, Ddpi_SavedStorage objects are not owned by Ddpi_StorageLocn objects, but by the Ddpi_Space object. A program analysis application can create multiple Ddpi_StorageLocn objects and Ddpi_SavedStorage objects.

Access to storage within a user address space is achieved through calls to the `ddpi_storagelocn_get_storage` and `ddpi_storagelocn_set_storage` operations. These operations use a Ddpi_StorageLocn object to identify the actual address space and location. Calls to these operations can invoke the Ddpi_Space_GS_Handler (get-storage handler) and Ddpi_Space_SS_Handler (save-storage handler) callback functions. These callback functions are the application-specific memory-access functions that were registered for the address space in the Ddpi_Space object).

Process flow: Accessing or modifying user storage

Access through the Ddpi_StorageLocn object can be either transparent (which allows specific elements within the space to be addressed individually) or opaque (which limits addressability to the space as a single element).

1. The program analysis application stores the user data in the application address space where the debugging objects, including Ddpi_Info and Ddpi_Space, were created.
2. The program analysis application calls the `ddpi_savedstorage_create` operation to create the appropriate Ddpi_SavedStorage objects. These objects describe each area of the user storage contained in the user storage space. They are created under the Ddpi_Space object that corresponds to the user address space. As each object is created, an identifier token is returned to the program analysis application.
3. The program analysis application can now use a Ddpi_StorageLocn object set in transparent mode to update the actual storage and insert the overlay hooks.
4. The program analysis application now interacts with the user or a command file and performs the required actions. Any user activity that requires access or modification of user storage will call `ddpi_storagelocn_get_storage` or `ddpi_storagelocn_set_storage`. This activity will use a Ddpi_StorageLocn object in opaque mode. The access operations check if any Ddpi_SavedStorage objects describe user storage for each request, and ensure that, for each address space, the appropriate application-specific access or update operation

© Copyright IBM Corp. 2004, 2009

will be invoked. Storage locations that are not identified by Ddpi_SavedStorage objects can be accessed by calling the get-storage and set-storage callback functions registered under the Ddpi_Space object.

5. When the user storage is restored, the `ddpi_savedstorage_get_modified` operation can be used to determine if the real storage should be updated.
6. When the user storage is terminated, the program analysis application must terminate the Ddpi_SavedStorage objects. The program analysis application calls the `ddpi_savedstorage_term` operation, passing the appropriate identifier token as an argument.

Transparent access

If the access policy in the `Ddpi_StorageLocn` object is set to transparent, calls to the `ddpi_storagelocn_get_storage` operation invoke the callback function contained in the `Ddpi_Space_GS_Handler` object and return the contents to the `Ddpi_StorageLocn` object. Similarly, calls to the `ddpi_storagelocn_set_storage` operation invoke the callback function contained in the `Ddpi_Space_SS_Handler` object and update the memory address in the `Ddpi_StorageLocn` object.

Opaque access

If the Ddpi_StorageLocn object specifies the opaque access policy, the program analysis (user) application can insert a saved storage block between the actual storage and the application. This allows the application to hide modifications to the real storage, such as illegal instructions that allow control to be caught.

The `Ddpi_SavedStorage` object represents the saved storage block. It is created by the `ddpi_savedstorage_create` operation. If this block does not exist for all or part of a storage read request, then the `Ddpi_Space_GS_Handler` callback function (provided when the `ddpi_space_create` operation is called) is used to access the `Ddpi_StorageLocn` object . Similarly, for a write request, if a saved storage block exists for a given address range, it is updated with the new data. Otherwise, the `Ddpi_Space_SS_Handler` callback function is called for data in addresses that do not correspond to any existing saved storage block.

A common use of opaque storage would be to create overlay hooks, where the first two bytes of the user-program instruction are saved and replaced by a service call. If the program analysis application dumps or disassembles storage containing those bytes, it calls the `ddpi_storagelocn_get_storage` operation in opaque mode, and the original opcode is revealed. The program analysis application can use a `Ddpi_StorageLocn` object with a transparent access policy to set up and remove the overlay hook.

Ddpi_SL_Policy_Trans object

This is an enumeration that is used to specify the storage access policy for a given `Ddpi_StorageLocn` object.

Type definition

```
typedef enum Ddpi_SL_Policy_s {
  Ddpi_SL_Policy_Unknown  = 0,
  Ddpi_SL_Policy_Opaque   = 1,
  Ddpi_SL_Policy_Trans    = 2
} Ddpi_SL_Policy;
```

Members

Ddpi_SL_Policy_Unknown
If this value is 0, the policy is unknown. Not a recommended setting.

Ddpi_SL_Policy_Opaque
If this value is 1, the `Ddpi_StorageLocn` access is opaque.

Ddpi_SL_Policy_Tran
If this value is 2, the `Ddpi_StorageLocn` is transparent.

Ddpi_SStor_Token object

The token of the saved storage block, defined as an opaque data type.

Type definition

```
typedef unsigned int    Ddpi_SStor_Token;
```

Ddpi_StorageLocn object

Ddpi_storagelocn is a pointer to data in memory, and not the actual data itself. It is an opaque data type that represents the address of a given location within an address space that is targeted by the user.

The Ddpi_storagelocn object contains:

- The address space, in the form of a “Ddpi_Space object” on page 33.
- The location address within the address space.
- The access policy, either opaque or transparent, to be used when accessing the storage.
- The user area, which is an optional object-extension area, and may be used by the Ddpi_StorageLocn object caller.

The creation, initialization, and destruction of a Ddpi_storagelocn object are handled as follows:

- The object is created by a successful call to the `ddpi_storagelocn_create` operation.
- The object is initialized by a successful call to the `ddpi_storagelocn_init` operation.
- Memory is deallocated by a successful call to the `ddpi_storagelocn_term` operation.

Type definition

```
typedef struct Ddpi_StorageLocn_s * Ddpi_StorageLocn;
```

Ddpi_SavedStorage object

The Ddpi_SavedStorage object, defined as an opaque data type, represents a block in the local-address space of an application.

This object contains a buffer that is to be used instead of the real memory at a given address. Ddpi_SavedStorage object contains:

- The starting address (low address) within the user address space to be replaced by the given buffer.
- The length of the user storage.

- The buffer of the user-address space.
- The modification flag, which if set, indicates that this storage has been modified since its creation.

A Ddpi_SavedStorage object is created by a successful call to ddpi_savedstorage_create. A Ddpi_SavedStorage object is freed by a call to the `ddpi_savedstorage_term` operation.

Type definition

```
typedef struct Ddpi_SavedStorage_s * Ddpi_SavedStorage;
```

ddpi_storagelocn_create operation

The `ddpi_storagelocn_create` operation creates a `Ddpi_StorageLocn` object. This object is a pointer to describe a location within a given address space (`Ddpi_Space`) and the access policy.

Prototype

```
int ddpi_storagelocn_create(
  Ddpi_Space            space,
  Dwarf_Addr            addr,
  Ddpi_SL_Policy        policy,
  int                   user_area_len,
  Ddpi_StorageLocn*     ret_locn,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object, which represents the address space that is pointed to by the `Ddpi_StorageLocn`.

addr
Input. This accepts the address in the address space pointed to by the `Ddpi_StorageLocn`. The address can be changed at any time prior to termination of the `Ddpi_StorageLocn` by using `ddpi_storagelocn_set_addr`.

policy
Input. This accepts the initial storage policy for this `Ddpi_StorageLocn`. The policy can be changed using `ddpi_storagelocn_set_policy`.

user_area_len
Input. This accepts the length of the user area.

ret_locn
Output. This returns the `Ddpi_StorageLocn` object created by this call.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the `Ddpi_StorageLocn` object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL

- ret_locn is NULL
- policy is an invalid policy
- There is insufficient memory

ddpi_storagelocn_term operation

The ddpi_storagelocn_term operation releases all the resources associated with the Ddpi_StorageLocn object.

Prototype

```
int    ddpi_storagelocn_term(
       Ddpi_StorageLocn    locn,
       Ddpi_Error *        error);
```

Parameters

locn
Input. This accepts the Ddpi_StorageLocn object to be freed.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
Returned upon successful release of all resources associated with the Ddpi_StorageLocn object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- locn is NULL
- An error occurs during memory deallocation.

ddpi_storagelocn_get_space operation

The ddpi_storagelocn_get_space operation returns the Ddpi_Space object referred to by the given Ddpi_StorageLocn object.

Prototype

```
int    ddpi_storagelocn_get_space(
       Ddpi_StorageLocn    locn,
       Ddpi_Space *        space,
       Ddpi_Error*         error);
```

Parameters

locn
Input. This accepts the Ddpi_StorageLocn object, which contains the desired Ddpi_Space.

space
Output. This returns the Ddpi_Space object referred to by the given Ddpi_StorageLocn object.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the Ddpi_Space object referred to by the given Ddpi_StorageLocn object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if an error occurs.

ddpi_storagelocn_set_space operation

The ddpi_storagelocn_set_space operation assigns the given address space as the Ddpi_StorageLocn object's new target address space.

Prototype

```
int    ddpi_storagelocn_set_space(
       Ddpi_StorageLocn    locn,
       Ddpi_Space          space,
       Ddpi_Error*         error);
```

Parameters

locn
Input. This accepts the Ddpi_StorageLocn object that is to be updated.

space
Input. This accepts the Ddpi_Space object to be assigned to the given Ddpi_StorageLocn. This value will override the previous space value.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the Ddpi_Space object referred to by the given Ddpi_StorageLocn object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- locn is NULL
- space is NULL
- locn does not have the correct version or is corrupted

ddpi_storagelocn_get_addr operation

The ddpi_storagelocn_get_addr operation examines the current Ddpi_StorageLocn object and returns the current address.

Prototype

```
int    ddpi_storagelocn_get_addr(
       Ddpi_StorageLocn    locn,
       Dwarf_Addr *        addr,
       Ddpi_Error*         error);
```

Parameters

locn
Input. This accepts the Ddpi_StorageLocn object that has the current address value.

addr
Output. This is where the address value from the Ddpi_StorageLocn object is returned.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the current address of the given Ddpi_StorageLocn object

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if there is an error.

ddpi_storagelocn_set_addr operation

The ddpi_storagelocn_set_addr operation changes the address in the given Ddpi_StorageLocn object to the given address.

The CDA user can then use this updated Ddpi_StorageLocn to access memory at the new address.

Prototype

```
int    ddpi_storagelocn_set_addr(
       Ddpi_StorageLocn    locn,
       Dwarf_Addr          addr,
       Ddpi_Error*         error);
```

Parameters

locn
Input. This accepts the Ddpi_StorageLocn object, which will be updated with the given address.

addr
Input. This accepts the new address for the given Ddpi_StorageLocn object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful update of the address of the given Ddpi_StorageLocn object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- locn is NULL.
- The Ddpi_Space object is NULL.
- addr is outside of the range of the Ddpi_Space object.
- locn does not have the correct version or is corrupted.

ddpi_storagelocn_get_storage operation

The ddpi_storagelocn_get_storage operation requests that bufferLn bytes of data be read from the address and address space of a Ddpi_StorageLocn object, then written to the data buffer in the caller's address space.

When writing, the ddpi_storagelocn_get_storage operation conforms to the Ddpi_StorageLocn access policy and any relevant Ddpi_SavedStorage objects.

In the absence of read access problems (such as page, access or segment exceptions), ddpi_storagelocn_get_storage reads bufferLn bytes storage associated with the Ddpi_StorageLocn object, returns it through buffer and sets dataLn to bufferLn. Otherwise, it sets dataLn to the actual data length read, and returns the read storage through buffer. This operation will use the get_storage_handler that was provided when creating the ddpi_space (unless a storagelocn_policy is opaque and the requested storage is in a saved storage block).

Prototype

```
int    ddpi_storagelocn_get_storage(
       Ddpi_StorageLocn    locn,
       char *              buffer,
       Dwarf_Unsigned      bufferLn,
       Dwarf_Unsigned*     ret_dataLn,
       Ddpi_Error*         error);
```

Parameters

locn
Input. This accepts the Ddpi_StorageLocn object that points to the location of the data that is to be read. This object supplies the address space, the address of the data, and the access policy.

buffer
Input/Output. This accepts and returns the buffer to which the requested storage will be written. Only the first ret_dataLn bytes of data will contain valid values on the return.

bufferLn
Input. This accepts the data length to read.

ret_dataLn
Output. This returns the actual data length read. It will return zero if no values were read. It will return a non-zero value if a portion was successfully read. In either case, the return code does not affect this value.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the pointer to the bufferLn bytes storage.

DW_DLV_NO_ENTRY
Returned if bufferLn is zero.

DW_DLV_ERROR
This value is returned if an error occurs.

ddpi_storagelocn_set_storage operation

The ddpi_storagelocn_set_storage operation requests that bufferLn bytes of data be written via the Ddpi_StorageLocn object from the data buffer in the caller's address space, respecting the Ddpi_StorageLocn access policy and any relevant Ddpi_SavedStorage objects.

In the absence of write access problems (such as page, access or segment exceptions), ddpi_storagelocn_set_storage writes bufferLn bytes of storage associated with the Ddpi_StorageLocn object from buffer and sets dataLn to bufferLn. Otherwise, it sets dataLn to the actual data length written. This operation will use the set_storage_handler that was provided when creating the ddpi_space (unless a storagelocn_policy is opaque and the requested storage is already mapped in a saved storage block).

Prototype

```
int    ddpi_storagelocn_set_storage(
       Ddpi_StorageLocn    locn,
       char *              buffer,
       Dwarf_Unsigned      bufferLn,
       Dwarf_Unsigned*     ret_dataLn,
       Ddpi_Error*         error);
```

Parameters

locn
Input. This accepts the Ddpi_StorageLocn object that points to the location where the data is to be written. This object supplies the address space, the address of the data, and the access policy.

buffer
Input. This accepts the buffer that contains the data to be written to the storage location.

bufferLn
Input. This accepts the number of bytes from the buffer that should be written to storage.

ret_dataLn
Output. This returns the number of bytes that were written to storage. This value should be zero if no bytes were written. This value should be non-zero if an error occurred when writing part of the data (even though the return code from this operation could still be DW_DLV_ERROR). It will return zero if no bytes were written. It will return a non-zero value if an error occurred while writing part of the data, even if the return code from this operation is DW_DLV_ERROR.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the pointer to the bufferLn bytes storage.

DW_DLV_NO_ENTRY
Returned if `bufferLn` is zero.

DW_DLV_ERROR
This value is returned if:

- `locn` is NULL.
- The Ddpi_Space object is NULL.
- The data to be written into the `Ddpi_StorageLocn` object exceeds the boundary of the `Ddpi_Space` object.
- `locn` does not have the correct version or is corrupted.

ddpi_storagelocn_get_policy operation

The `ddpi_storagelocn_get_policy` operation returns the policy of the given `Ddpi_StorageLocn` object.

Prototype

```
int    ddpi_storagelocn_get_policy(
       Ddpi_StorageLocn    locn,
       Ddpi_SL_Policy*     return_policy,
       Ddpi_Error*         error);
```

Parameters

locn
Input. This accepts the `Ddpi_StorageLocn` object to be queried for its current policy setting.

return_policy
Output. This is the location where the operation returns the access policy (opaque or transparent) of the given `Ddpi_StorageLocn` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the pointer to the policy of the given `Ddpi_StorageLocn` object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `locn` is NULL.
- The `Ddpi_Space` object is NULL.
- `return_policy` is NULL.
- `locn` does not have the correct version or is corrupted.

ddpi_storagelocn_set_policy operation

The `ddpi_storagelocn_set_policy` operation updates the given `Ddpi_StorageLocn` object with the given policy.

Prototype

```
int    ddpi_storagelocn_set_policy(
       Ddpi_StorageLocn    locn,
       Ddpi_SL_Policy      policy,
       Ddpi_Error*         error);
```

Parameters

locn
Input. This accepts the `Ddpi_StorageLocn` object to be updated.

policy
Input. This accepts the access policy value that will replace the current policy of the `Ddpi_StorageLocn` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful update of the policy of the given `Ddpi_StorageLocn` object.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `locn` is NULL.
- The `Ddpi_Space` object is NULL.
- `policy` is invalid.
- `locn` does not have the correct version or is corrupted.

ddpi_storagelocn_get_user_area operation

The `ddpi_storagelocn_get_user_area` operation returns the address of the user area for the given `Ddpi_StorageLocn` object.

Prototype

```
int    ddpi_storagelocn_get_user_area(
       Ddpi_StorageLocn    locn,
       char **             return_user_area,
       Ddpi_Error *        error);
```

Parameters

locn
Input. This accepts the `Ddpi_StorageLocn` object.

return_user_area
Output. This returns the address of the user area for the given `Ddpi_StorageLocn` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the pointer to the user area.

DW_DLV_NO_ENTRY
Returned if the user area length is zero.

DW_DLV_ERROR
This value is returned if:

- `locn` is NULL.
- The `Ddpi_Space` object is NULL.
- `locn` does not have the correct version or is corrupted.

ddpi_savedstorage_create operation

The `ddpi_savedstorage_create` operation creates a `Ddpi_SavedStorage` object to identify substituted user storage.

The contents of the given buffer are copied into `libddpi` storage, and then stored so that they can quickly be accessed by the address. This operation returns a token, which identifies a specific saved storage area through the `ret_token` parameter.

Prototype

```
int    ddpi_savedstorage_create(
       Ddpi_Space       space,
       Dwarf_Addr       locn,
       unsigned int     len,
       Dwarf_Ptr buffer buffer,
       Ddpi_SStor_Token ret_token,
       Ddpi_Error*      error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

locn
Input. This accepts the starting address of the given address space.

len
Input. This accepts the length of the given address space.

buffer
Input. This accepts the content of the given address space.

ret_token
Output. This returns the token used to identify the `Ddpi_SavedStorage` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the token that identifies the saved storage area.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL.

- buffer is zero in length.
- ret_token is NULL.
- space exceeds the limit of the Ddpi_Space object.
- ret_token does not have the correct version or is corrupted.
- There is insufficient memory to create a Ddpi_SavedStorage object.

ddpi_savedstorage_term operation

The ddpi_savedstorage_term operation releases all the resources associated with the given Ddpi_SavedStorage object.

Prototype

```
int    ddpi_savedstorage_term(
       Ddpi_Space        space,
       Ddpi_SStor_Token  token,
       Ddpi_Error*       error);
```

Parameters

space
Input/Output. This accepts and returns the Ddpi_Space object that contains the given Ddpi_SavedStorage token.

token
Input. This accepts the Ddpi_SavedStorage object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the token that identifies the saved storage area.

DW_DLV_NO_ENTRY
Returned if the token is invalid.

DW_DLV_ERROR
This value is returned if:

- space is NULL.
- The Ddpi_SavedStorage object does not have the correct version or is corrupted.
- An error occurs during deallocation of memory.

ddpi_savedstorage_term_all operation

The ddpi_savedstorage_term_all operation releases all the resources associated with all of the Ddpi_SavedStorage objects attached to a given space.

Prototype

```
int ddpi_savedstorage_term_all(
      Ddpi_Space      space,
      Ddpi_Error*     error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful release of the resources associated with all the saved storage objects attached to the given storage space.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL
- Any of the Ddpi_SavedStorage objects associated with `space` do not have the correct version or is corrupted
- An error occurs during deallocation of memory.

ddpi_savedstorage_list_all_tokens operation

The `ddpi_savedstorage_list_all_tokens` operation returns a list of all the saved storage tokens in a given `Ddpi_Space` object.

This list must be freed by the caller with the following code:

```
ddpi_dealloc(info, *ret_tokens, DDPI_DLA_SSTOR_TOKEN)
```

Prototype

```
int ddpi_savedstorage_list_all_tokens(
    Ddpi_Space          space,
    Ddpi_SStor_Token**  ret_tokens,
    Dwarf_Unsigned*     num_tokens,
    Ddpi_Error*         error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

ret_tokens
Output. This returns the list of the tokens in the `Ddpi_Space` object.

num_tokens
Output. This returns the number of tokens in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the list of saved storage tokens.

DW_DLV_NO_ENTRY
Returned if there are no saved storage tokens in `space`.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL
- Any of the Ddpi_SavedStorage objects associated with `space` do not have the correct version or is corrupted
- An error occurs during deallocation of memory.

ddpi_savedstorage_list_modified operation

The `ddpi_savedstorage_list_modified` operation returns a list of all the saved storage tokens in a given `Ddpi_Space` that have the modified bits set.

This list must be freed by the caller with the following code:

```
ddpi_dealloc(info, *ret_tokens, DDPI_DLA_SSTOR_TOKEN)
```

Prototype

```
int ddpi_savedstorage_list_modified(
    Ddpi_Space          space,
    Ddpi_SStor_Token**  ret_tokens,
    Dwarf_Unsigned*     num_tokens,
    Ddpi_Error*         error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

ret_tokens
Output. This returns the list of the modified tokens in the `Ddpi_Space` object.

num_tokens
Output. This returns the number of tokens in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the list of all the saved storage tokens, in the given Ddpi_Space object, that have the modified bits set.

DW_DLV_NO_ENTRY
Returned if, in `space`, there is no saved storage token with a modified bit set.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL
- Any of the Ddpi_SavedStorage objects associated with `space` do not have the correct version or is corrupted
- An error occurs during deallocation of memory.

ddpi_savedstorage_get operation

The ddpi_savedstorage_get operation queries the Ddpi_SavedStorage object that corresponds to the given token.

The ddpi_savedstorage_get operation returns:

- The start address of the saved storage block (ret_locn)
- The size of the saved storage block (ret_len)
- The contents of the saved storage block (ret_buffer)

The consumer must not delete or modify the storage pointed to by ret_buffer.

Prototype

```
int ddpi_savedstorage_get(
  Ddpi_Space            space,
  Ddpi_SStor_Token      token,
  Dwarf_Addr*           ret_locn,
  unsigned int*         ret_len,
  Dwarf_Ptr*            ret_buffer,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the Ddpi_Space object. This is the address space that contains the saved storage block to be queried.

token
Input. This accepts the Ddpi_SavedStorage token to be queried.

ret_locn
Output. This is a pointer to an address. The address gives the location of the saved storage buffer in its parent address space. This field cannot be NULL.

ret_len
Output. This is a pointer to the location where the length of the saved storage block will be returned. This field cannot be NULL.

ret_buffer
Output. This is a pointer to the buffer where the contents of the saved storage block will be returned. This is the libddpi internal storage block and should not be modified or deallocated by the user. This field cannot be NULL.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of:

- The start address of the saved storage block (ret_locn).
- The size of the saved storage block (ret_len).
- The contents of the saved storage block (ret_buffer).

DW_DLV_NO_ENTRY
Returned if there is not a saved storage block corresponding to the given token.

DW_DLV_ERROR
This value is returned if:

- space is NULL
- ret_buffer is NULL
- One of the Ddpi_SavedStorage objects associated with space do not have the correct version or is corrupted

ddpi_savedstorage_list_all operation

The ddpi_savedstorage_list_all operation returns a list of all the saved-storage tokens in a given space.

Prototype

```
int ddpi_savedstorage_list_all(
    Ddpi_Space          space,
    Ddpi_SStor_Token**  ret_tokens,
    Dwarf_Unsigned*     num_tokens,
    Ddpi_Error*         error);
```

Parameters

space
Input. This accepts the Ddpi_Space object.

ret_tokens
Output. This returns the list of tokens.

num_tokens
Output. This returns the number of tokens in the list.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the list of all the saved-storage tokens in the given space.

DW_DLV_NO_ENTRY
Returned if there are no saved-storage tokens in the space.

DW_DLV_ERROR
This value is returned if:

- space and its associated Ddpi_Info object is NULL.
- ret_tokens is NULL.
- num_tokens is NULL.

ddpi_savedstorage_find operation

The ddpi_savedstorage_find operation finds the first Ddpi_SavedStorage token that identifies an address range in a given space.

The first token maps all or part of the given location and length within the given address space.

Notice that a given range can have multiple Ddpi_SavedStorage objects.

This operation returns only the first token. To find the next token, use ddpi_savedstorage_next.

To find the address range and contents that correspond to the returned token, use ddpi_savedstorage_get.

Prototype

```
int ddpi_savedstorage_find(
  Ddpi_Space            space,
  Dwarf_Addr            locn,
  unsigned int          len,
  Ddpi_SStor_Token*     ret_token,

  Ddpi_Error*           error);
```

Parameters

space
Input/Output. This accepts and returns the Ddpi_Space object. This represents the address space that might have a saved storage block at the given location.

locn
Input. This accepts the address location. This is the start of the address range, in which the operation will look for a saved storage block.

len
Input. This defines the length of the area of memory that starts at locn and runs until locn + len - 1. The zone of memory defined by this range is where this operation will search to see if there is a saved storage block.

ret_token
Output. This returns the token that corresponds to the first saved storage block in the given address range.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the first token that identifies an address range in a given space.

DW_DLV_NO_ENTRY
Returned if there is no Ddpi_SavedStorage object at the location with the address space.

DW_DLV_ERROR
This value is returned if:

- space is NULL
- ret_token is NULL
- len is zero in length
- One of the Ddpi_SavedStorage objects associated with space does not have the correct version or is corrupted

ddpi_savedstorage_next operation

The ddpi_savedstorage_next operation finds the Ddpi_SavedStorage token following the given token.

This is defined as the storage block that follows the given token with the closest possible address.

Prototype

```
int ddpi_savedstorage_next(
  Ddpi_Space            space,
  Ddpi_SStor_Token      token,
  Ddpi_SStor_Token*     ret_next_token,

  Ddpi_Error*           error);
```

Parameters

space
Input. This is the address space that holds the given saved storage token (`token`).

token
Input. A token for a saved storage block that exists in the given space.

ret_next_token
Output. This parameter returns the address of the next token.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the Ddpi_SavedStorage token following the given token.

DW_DLV_NO_ENTRY
Returned if there is no Ddpi_SavedStorage object at the location with the address space.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL.
- `token` does not refer to an existing saved-storage block.
- `ret_next_token` is NULL.
- One of the Ddpi_SavedStorage objects associated with `space` does not have the correct version or is corrupted.

ddpi_savedstorage_get_modified operation

The `ddpi_savedstorage_get_modified` operation queries the `Ddpi_SavedStorage` buffer, and indicates if it has been modified, either by a `ddpi_storagelocn_set_storage` to the saved storage block, or forced by a call to `ddpi_savedstorage_set_modified`.

Prototype

```
int ddpi_savedstorage_get_modified(
  Ddpi_Space            space,
  Ddpi_SStor_Token      token,
  Dwarf_Bool*           ret_bool,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object to which the saved storage block belongs.

token
Input. This accepts the `Ddpi_SavedStorage` token that corresponds to the relevant saved storage area.

ret_bool
Output. A pointer to a boolean value. This will be true if the call to `ddpi_storagelocn_set_storage` has occurred to the saved storage block, or if the CDA user has explicitly set the modified bit by calling `ddpi_saved_storage_set_modified`.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the boolean value that indicates whether the storage buffer has been modified.

DW_DLV_NO_ENTRY
Returned if there is no saved storage block that corresponds to the given token.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL.
- `ret_bool` is NULL
- One of the Ddpi_SavedStorage objects associated with `space` does not have the correct version or is corrupted.

ddpi_savedstorage_set_modified operation

The `ddpi_savedstorage_set_modified` operation sets the modification bit for the given saved storage token for the given `Ddpi_Space` object.

Prototype

```
int ddpi_savedstorage_set_modified(
  Ddpi_Space            space,
  Ddpi_SStor_Token      token,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the address space that contains the saved storage block that will be marked as modified.

token
Input. This accepts the token that corresponds to the saved storage block that is to be marked as modified.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
Returned when the modification bit for the given saved storage token for the given Ddpi_Space object has been successfully reset.

DW_DLV_NO_ENTRY
Returned if there is no saved storage block that corresponds to the given token.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL.
- `token` is NULL.
- One of the Ddpi_SavedStorage objects associated with `space` does not have the correct version or is corrupted.

ddpi_savedstorage_reset_change operation

The `ddpi_savedstorage_reset_change` operation sets the modified bit of the saved storage block identified by the given token in the given `Ddpi_Space`.

Prototype

```
int ddpi_savedstorage_reset_change(
  Ddpi_Space            space,
  Ddpi_SStor_Token      token,
  Ddpi_Error*           error);
```

Parameters

space
Input. This accepts the address space that contains the saved storage block to be marked as not modified.

token
Input. This accepts the token that identifies the saved storage block to be marked as not modified. This token must refer to an active saved storage block.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the modified bit in the given Ddpi_Space object has been successfully reset.

DW_DLV_NO_ENTRY
Returned if there is no saved storage block that corresponds to the given token.

DW_DLV_ERROR
This value is returned if an error occurs.

ddpi_savedstorage_dump operation

The `ddpi_savedstorage_dump` operation dumps out the saved storage blocks (token, address, length and contents) to the given `FILE`.

`ddpi_savedstorage_dump` is provided to aid the consumer in debugging their application.

Prototype

```
int ddpi_savedstorage_dump(
  Ddpi_Space            space,

  FILE*                 fp,
  Dwarf_Bool            Ascii,
  Ddpi_Error*           error)
```

Parameters

space
Input. This accepts the `Ddpi_Space` object to which the saved storage block corresponds.

fp Input. This accepts the destination for the dumped information.

Ascii
Input. This accepts the type of character set for the information: ASCII (1), or EBCDIC (0).

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the contents of the saved storage blocks (token, address, length and contents) has been successfully dumped to a file

DW_DLV_NO_ENTRY
Returned if there is no saved storage block.

DW_DLV_ERROR
This value is returned if:

- `space` is NULL.
- `fp` is NULL.
- One of the Ddpi_SavedStorage objects associated with `space` does not have the correct version or is corrupted.
- There is insufficient memory.

Chapter 30. Ddpi_Format APIs

The Ddpi_Format APIs control the content and format of the DWARF debugging information that is sent to output. The APIs use a combination of standard printf specifiers and Ddpi_Format specifiers. The ddpi_format operations generate DWARF debugging information; some of the operations override the default behavior of the Ddpi_Format specifiers.

The Ddpi_Format APIs support all data types, including complex types that contain different values or variables. Complex types include structures, unions, classes, enumerated types, arrays, and functions, also referred to as *composite types*.

Note: If the module map is used, the access field in the Ddpi_Xeval_Context object must be set before is passed into any of the ddpi_format operations.

For your convenience, some operations replace the use of Ddpi_Format specifiers.

Ddpi_Format specifiers

The libddpi design recognizes that output data will include the names and values of variables, types and functions. The Ddpi_Format specifiers handle this type of information.

For example:

- The %<NV> specifier indicates that a variable name should be sent to output.
- The %<I> specifier indicates that an indentation should be added, based on the nested position of the given data.

Composite-type data can be extracted for an entire structure or for a single member. For example, if theddpi_formatter operation specifies %<Nd> , the function will return both the tag name and the variable name of the given composite variable.

Specifiers that identify output data

%<NT>
%<NT> causes the function to return the type of the variable.

%<NV>
%<NV> causes the function to return the name of the variable.

%<Nd>
%<Nd> causes the function to return both the type and name of the variable.

%<ND>
%<ND> behaves the same as %<Nd>, except that the composite types are expanded. For example, if a structure is used, then the function will also return the type and name of the members.

%<V>
%<V> causes the function to return the value of the variable.

Specifiers that format output data

%<Ix>
Causes the function to create an indentation, where x is the depth in the current

© Copyright IBM Corp. 2004, 2009

nested level of a composite type. For example, %<I3> will cause an indentation of three spaces per level. For a member of a structure inside a structure, the indentation would be six spaces.

%<S>
Causes the function to print a string that is enclosed in double quotes, and replace non-printable characters with dots.

%<C>
Causes the function to print a character enclosed in single quotes. It replaces control characters, based on the input character set (EBCDIC or ASCII), as shown in Table 3. It replaces other non-printing characters with dots.

Table 3. Control character replacements for ASCII and EBCDIC

EBCDIC input character	ASCII input character	Replacement string
\x15	\xa	\n
\x5	\x9	\t
\xb	\xb	\v
\x16	\x8	\b
\xd	\xd	\r
\xc	\xc	\f
\x2f	\x7	\a

user defined
The format specifier also accepts the general syntax %<fmt_name> , where fmt_name is a symbolic name representing a format string. fmt_name must be defined by a previous call to the ddpi_format_set_type_format operation. Substitution is done recursively until all items with the %<...> syntax are replaced.

Example: If you want to replace all occurrences of %<UINT> in the given format string with %12.12u, use the following statement:

```
ddpi_format_set_type_format(info, "UINT", "%12.12u");
```

ddpi_formatter operation

The ddpi_formatter operation formats the data for a given DIE according to the specifiers in a format string.

For example, the following is called:

```
ddpi_formatter(info, context, die, "The %<NT> variable %<NV> has value
%<V>", &b, &err);
```

The DIE represents the variable with the declaration int i = 2;. This will return the string:

```
The int variable i has value 2.
```

The formatted string is returned in return_buff. The caller should free the storage using dwarf_dealloc with type DW_DLA_STRING.

Prototype

```
int ddpi_formatter(
  Ddpi_Info             info,
  Ddpi_Xeval_Context*   context,
  Dwarf_Die             die,
  char*                 fmt,
  char**                return_buff,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

context
Input. If a module map is used, this accepts the `Ddpi_Xeval_Context` array. The `access` field in the `Ddpi_Xeval_Context` object must be set before is passed into any of the `ddpi_format` operations.

die
Input. This accepts the DIE to be formatted.

fmt
Input. This accepts the format string.

return_buff
Output. This returns the formatted string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful format of the data for a given DIE.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `info`, `context`, `die` or `return_buff` is NULL.
- Any other internal error occurs. (Check the `Ddpi_Error` object for the cause.)

ddpi_format_address operation

The `ddpi_format_address` operation formats the data at the address in the given token according to the type DIE. It returns the formatted string in `return_buff`.

The formatting is done as `%<V>`. (See “Ddpi_Format specifiers” on page 295). Array subranges can also be formatted.

The caller should free the memory allocated for the return string using `ddpi_dealloc` with type `DDPI_DLA_STRING`.

Prototype

```
int ddpi_format_address(
  Ddpi_Info             info,
  Ddpi_Xeval_Context*   context,
  Ddpi_Xeval_Token      token,
  Dwarf_Die             type_die,
```

```
Dwarf_Unsigned        low_index,
Dwarf_Unsigned        count,
char**                return_buff,
Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

context
Input. This accepts the `Ddpi_Xeval_Context` array. The `access` field in the Ddpi_Xeval_Context object must be set before is passed into any of the `ddpi_format` operations.

token
Input. If a module map is used, this accepts the XEVAL token.

type_die
Input. This accepts the type DIE.

low_index
Input. This accepts the low index in the subrange if the data is an array. Otherwise this parameter is ignored.

count
Input. This accepts the number of elements to format if the data is an array. If the value is 0, then the whole array will be formatted.

return_buff
Output. This returns the formatted string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful call of the address data for a given DIE.

DW_DLV_NO_ENTRY
Returned if the token is not an address.

DW_DLV_ERROR
This value is returned if:

- `info`, `context`, `type_die`, or `return_buff` is NULL.
- Any other internal error occurs. (Check the `Ddpi_Error` object for the cause.)

ddpi_format_bitfield_address operation

The `ddpi_format_bitfield_address` operation formats the bit field from the member token and returns the formatted string in `return_buff`.

The formatting is done as `%<V>`. (See “Ddpi_Format specifiers” on page 295). Array subranges can also be formatted.

The caller should free the memory allocated for the return string using `ddpi_dealloc` with type `DDPI_DLA_STRING`.

Prototype

```
int ddpi_format_bitfield_address(
  Ddpi_Info           info,
  Ddpi_Xeval_Context* context,
  Dwarf_Die           member_die,
  Ddpi_Xeval_Token    member_token,
  char**              return_buff,
  Ddpi_Error*         error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

context
Input. If a module map is used, this accepts the `Ddpi_Xeval_Context` array. The `access` field in the Ddpi_Xeval_Context object must be set before is passed into any of the `ddpi_format` operations.

member_die
Input. This accepts the bitfield member DIE.

member_token
Input. This accepts the member token from XEVAL.

return_buff
Output. This returns the formatted string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful format of the bit field address.

DW_DLV_NO_ENTRY
Returned if the DIE is not for a bit field.

DW_DLV_ERROR
This value is returned if:

- `info` is NULL
- `context`, `member_die`, or `return_buff` is NULL
- An error occurs

ddpi_format_set_input_charset operation

The `ddpi_format_set_input_charset` operation assigns the character set for characters that are passed to the formatter.

Prototype

```
int ddpi_format_set_input_charset(
  Dwarf_IBM_charset_type cs);
```

Parameters

cs Input. This accepts the character-set type.

Return values

The `ddpi_format_set_input_charset` operation always returns `DW_DLV_OK`.

ddpi_format_set_type_format operation

The ddpi_format_set_type_format operation assigns the type format specifier for the given type name.

Prototype

```
int ddpi_format_set_type_format(
  Ddpi_Info             info,
  char*                 typeName,
  char*                 fmt,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the Ddpi_Info object.

typeName
Input. This accepts the name of the type.

fmt
Input. This accepts the format specifier for the type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the type format specifier for the given type name.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- info is NULL.
- typeName is NULL.
- fmt is NULL.
- An error occurs while allocating memory.

ddpi_format_get_type_format operation

The ddpi_format_get_type_format operation returns the type format specifier for the given type name.

Prototype

```
int ddpi_format_get_type_format(
  Ddpi_Info             info,
  char*                 name,
  char**                ret_format,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the Ddpi_Info object.

typeName
Input. This accepts the name of the type.

fmt
Output. This returns the format specifier for the type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the type format specifier for the given type name.

DW_DLV_NO_ENTRY
Returned if the type name is not found.

DW_DLV_ERROR
This value is returned if `ret_fmt` is NULL.

ddpi_format_set_composite_format operation

The `ddpi_format_set_composite_format` operation assigns the composite format specifiers for the given composite type name.

Prototype

```
int ddpi_format_set_composite_format(
  Ddpi_Info             info,
  char*                 typeName,
  char*                 head,
  char*                 body,
  char*                 aux,
  char*                 tail,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

typeName
Input. This accepts the name of the type.

head
Input. This accepts the returned format specifier for the head.

body
Input. This accepts the returned format specifier for the body.

aux
Input. This accepts the returned format specifier for the auxiliary separator.

tail
Input. This accepts the returned format specifier for the tail.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the composite format specifiers for the given composite type name.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `info` is NULL
- `typeName` is NULL
- `head`, `body`, `aux`, or `tail` is NULL
- An error occurs during memory allocation.

ddpi_format_get_composite_format operation

The `ddpi_format_get_composite_format` operation returns the composite format specifiers for the given composite type name.

Prototype

```
int ddpi_format_get_composite_format(
  Ddpi_Info             info,
  char*                 name,
  char**                ret_head,
  char**                ret_body,
  char**                ret_aux,
  char**                ret_tail,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

name
Input. This accepts the name of the type.

ret_head
Output. This returns the format specifier for the head.

ret_body
Output. This returns the format specifier for the body.

ret_aux
Output. This returns the format specifier for the auxiliary separator.

ret_tail
Output. This returns the format specifier for the tail.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the composite format specifiers for the given composite type.

DW_DLV_NO_ENTRY
Returned if the type name is not found.

DW_DLV_ERROR
This value is returned if:

- `ret_head`
- `ret_body`
- `ret_aux`
- `ret_tail`

ddpi_format_expand_type_format operation

The `ddpi_format_expand_type_format` operation returns the expanded format specifier string.

Prototype

```
int ddpi_format_expand_type_format(
  Ddpi_Info             info,
  char*                 fmt,
  char**                ret_buff,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

fmt
Input. This accepts the input specifier string.

ret_buff
Output. This returns the expanded specifier string.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the expanded format specifier.

DW_DLV_NO_ENTRY
Returned if the type name is not found.

DW_DLV_ERROR
This value is returned if:

- There is an error during memory allocation.
- `info` is NULL.
- `fmt` or `ret_buff` is NULL.

ddpi_format_i_to_hex operation

The `ddpi_format_i_to_hex` operation formats the value in hexadecimal format, given the size of the value.

Prototype

```
int ddpi_format_i_to_hex (
  Ddpi_Info             info,
  Dwarf_Unsigned        val,
  Dwarf_Unsigned        bytes,
  char**                ret_buff,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

val
Input. This accepts the value to be formatted.

bytes
Input. This accepts the size of the `val` in bytes.

ret_buff
Output. This returns the formatted output.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the value is successfully formatted in hexadecimal format.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- There is an error during memory allocation.
- `info` is NULL.
- `ret_buff` is NULL.

The `ddpi_format_i_to_hex` operation returns `DW_DLV_OK` The `ddpi_format_i_to_hex` operation returns `DW_DLV_ERROR` if:

- An error occurs while allocating memory

Note: `ddpi_format_i_to_hex` never returns `DW_DLV_NO_ENTRY`.

ddpi_format_hexdump operation

The `ddpi_format_hexdump` operation formats an area of storage in hexadecimal format according to the parameters given.

`ddpi_format_hexdump` places the content into columns, with each column displaying the given number of bytes in hexadecimal format.

Prototype

```
int ddpi_format_hexdump(
  Ddpi_Info             info,
  Dwarf_Ptr             area,
  Dwarf_Unsigned        arealen,
  Dwarf_Unsigned        col,
  Dwarf_Unsigned        col_bytes,
  char*                 delimiter,
  char**                ret_buff,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

area
Input. This accepts the pointer to the storage area that will be dumped.

arealen
Input. This accepts the length of the storage area that will be dumped.

col
Input. This accepts the number of columns per row.

col_bytes
Input. This accepts the number of bytes per column.

delimiter
Input. This accepts the column delimiter. This is NULL if the parameter is not used.

ret_buff
Output. This returns the formatted output.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the given area of storage is successfully formatted in hexadecimal format.

DW_DLV_NO_ENTRY
Returned if `arealen` is 0 or `area` is NULL.

DW_DLV_ERROR
This value is returned if:

- There is an error during memory allocation.
- `info` is NULL
- `ret_buff` is NULL
- `col` or `col_bytes` is 0

ddpi_format_chardump operation

The `ddpi_format_chardump` operation formats an area of storage in character format according to the parameters given.

`ddpi_format_chardump` places the content into columns, with each column displaying the given number of bytes in character format.

Prototype

```
int ddpi_format_chardump(
  Ddpi_Info             info,
  Dwarf_Ptr             area,
  Dwarf_Unsigned        arealen,
  Dwarf_Unsigned        col,
  Dwarf_Unsigned        col_bytes,
  char*                 delimiter,
  char*                 subchar,
  char**                ret_buff,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

area
Input. This accepts the pointer to the storage area that will be dumped.

arealen
Input. This accepts the length of the storage area that will be dumped.

col
Input. This accepts the number of columns per row.

col_bytes
Input. This accepts the number of bytes per column.

delimiter
Input. This accepts the column delimiter. This is NULL if the parameter is not used.

sub_char
Input. This accepts the substitution character for non-printing characters. This is NULL if the parameter is not used.

ret_buff
Output. This returns the formatted output.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the given area of storage is successfully formatted in character format.

DW_DLV_NO_ENTRY
Returned if `arealen` is 0 or `area` is NULL.

DW_DLV_ERROR
This value is returned if:

- There is an error during memory allocation.
- `info` is NULL.
- `ret_buff` is NULL.
- `col` or `col_bytes` is 0.

ddpi_format_dbx_hexdump operation

The `ddpi_format_dbx_hexdump` operation formats an area of storage in `dbx` style.

Prototype

```
int ddpi_format_dbx_hexdump(
  Ddpi_Info              info,
  Dwarf_Ptr              area,
  Dwarf_Signed           arealen,
  Dwarf_Bool             showsame,
  char**                 ret_buff,
  Ddpi_Error*            error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

area
Input. This accepts the pointer to the storage area that will be dumped.

arealen
Input. This accepts the length of the storage area that will be dumped.

showsame
Input. This accepts the option to display rows that are identical to the previous row.

ret_buff
Output. This returns the formatted output.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the given area of storage is successfully formatted in `dbx` style.

DW_DLV_NO_ENTRY
Returned if `arealen` is 0 or `area` is NULL.

DW_DLV_ERROR
This value is returned if:

- There is an error during memory allocation.
- `info` is NULL.
- `ret_buff` is NULL.
- `col` or `col_bytes` is 0.

ddpi_format_get_DIE_xeval_token operation

The `ddpi_format_get_DIE_xeval_token` operation returns the XEVAL token associated with the location expression of a given DIE with a `DW_AT_location` attribute.

The token entity must be freed by the caller as follows:

```
ddpi_dealloc(info, ret_token->xt_token, DDPI_DLA_CHUNK)
```

Prototype

```
int ddpi_format_get_DIE_xeval_token (
  Ddpi_Info             info,
  Ddpi_Xeval_Context*   context,
  Dwarf_Die              die,
  Ddpi_Xeval_Token*      ret_token,
  Ddpi_Error*            error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

context
Input. If a module map is used, this accepts the `Ddpi_Xeval_Context` array. The `access` field in the `Ddpi_Xeval_Context` object must be set before is passed into any of the `ddpi_format` operations.

die
Input. This accepts the DIE with the location expression.

ret_token
Output. This returns the token.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the XEVAL token associated with the location expression of a given DIE with a `DW_AT_location` attribute has been successfully returned.

DW_DLV_NO_ENTRY
Returned if the location expression does not exist.

DW_DLV_ERROR
This value is returned if:

- `info`, `context`, `die`, or `ret_token` is NULL.
- the module or machine state associated with `context` is NULL.
- Any other internal error occurs (check `Ddpi_Error` for the cause).

ddpi_format_get_DIE_member operation

The `ddpi_format_get_DIE_member` operation returns the member DIE (`DW_TAG_member`) and XEVAL token for a given member name within a given `DW_TAG_structure_type`, `DW_TAG_union_type`, or `DW_TAG_class_type` DIE.

Optionally, the location expression token can be retrieved if the returned XEVAL token is not NULL.

The returned DIE and XEVAL token must be freed by the caller as follows:

```
dwarf_dealloc (dbg, *ret_die, DW_DLA_DIE)
ddpi_dealloc(info, ret_token->xt_token, DDPI_DLA_CHUNK)
```

Prototype

```
int ddpi_format_get_DIE_member (
  Ddpi_Info              info,
  Ddpi_Xeval_Context*    context,
  Dwarf_Die              csu_type_die,
  Ddpi_Xeval_Token       die_token,
  char*                  memname,
  Dwarf_Die*             ret_die,
  Ddpi_Xeval_Token*      ret_token,
  Ddpi_Error*            error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

context
Input. If a module map is used, this accepts the `Ddpi_Xeval_Context` array. The `access` field in the `Ddpi_Xeval_Context` object must be set before is passed into any of the `ddpi_format` operations.

csu_type_die
Input. This accepts the `DW_TAG_class_type`, the `DW_TAG_structure_type`, or the `DW_TAG_union_type` DIE.

die_token
Input. This accepts the XEVAL token referencing the class, structure, or union object.

memname
Input. This accepts the member name to find.

ret_die
Output. This returns the `DW_TAG_member` DIE.

ret_token
Output. This returns the XEVAL token, or NULL if it is not used.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of both of the following:

- The member DIE (`DW_TAG_member`)
- The XEVAL token for a given member name within a given `DW_TAG_structure_type`, `DW_TAG_union_type`, or `DW_TAG_class_type` DIE

DW_DLV_NO_ENTRY
Returned if the member name cannot be found or if the location expression does not exist.

DW_DLV_ERROR
This value is returned if:

- `info`, `context`, `csu_type_die`, or `ret_die` is NULL.
- The module or machine state associated with `context` is NULL.
- Any other internal error occurs (check `Ddpi_Error` for the cause).

ddpi_format_get_array_DIE_xeval_token operation

The `ddpi_format_get_array_DIE_xeval_token` operation returns the location expression token for an array index and the type of DIE of the array.

The calling function should free the returned DIE and XEVAL token as follows:

```
dwarf_dealloc (dbg, *ret_die, DW_DLA_DIE)
ddpi_dealloc(info, ret_token->xt_token, DDPI_DLA_CHUNK)
```

Prototype

```
int ddpi_format_get_array_DIE_xeval_token (
  Ddpi_Info             info,
  Ddpi_Xeval_Context*   context,
  Dwarf_Die              array_die,
  Ddpi_Xeval_Token       die_token,
  Dwarf_Unsigned         index,
```

```
Dwarf_Bool              check_bound,
Ddpi_Xeval_Token*       ret_token,
Dwarf_Die*              ret_typedie,
Ddpi_Error*             error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

context
Input. If a module map is used, this accepts the `Ddpi_Xeval_Context` array. The `access` field in the Ddpi_Xeval_Context object must be set before is passed into any of the `ddpi_format` operations.

array_die
Input. This accepts the array DIE. Note that this must be a `DW_TAG_array_type` DIE.

die_token
Input. This accepts the XEVAL token referencing the array object.

index
Input. This accepts the index into the array.

check_bound
Input. This accepts the setting for the boundary flag of the array.

ret_token
Output. This returns the XEVAL token for the array element at the given index.

ret_typedie
Output. This returns the type DIE of the array at the given index. This must be must be a `DW_TAG_array_type` DIE.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of both of the following:

- The location expression token for an array index
- The type of DIE of the array

DW_DLV_NO_ENTRY
Returned if both of the following are true:

- `check_bound` is true
- `index` is out of range

DW_DLV_ERROR
This value is returned if:

- `info`, `context`, `ret_token`, or `ret_type_die` is NULL.
- `array_die` is NULL or not a `DW_TAG_array_type`.
- The module or machine state associated with `context` is NULL .
- A memory allocation error occurs.
- Any other internal error occurs (check `Ddpi_Error` for the cause).

ddpi_format_get_ptr_to_mem_xeval_token operation

The ddpi_format_get_ptr_to_mem_xeval_token operation returns an XEVAL token for a pointer to member.

To return an XEVAL token for a pointer to member, the ddpi_format_get_ptr_to_mem_xeval_token operation performs the following steps:

- Push the address of the pointer to member onto the location expression stack.
- Push the base address of the object onto the location expression stack.
- Evaluate the DW_AT_use_location expression for the pointer to the member type.

The returned token must be freed by the calling function as follows:

```
ddpi_dealloc(info, ret_this_token->xt_token, DDPI_DLA_CHUNK)
ddpi_dealloc(info, ret_mbr_token->xt_token, DDPI_DLA_CHUNK)
```

Prototype

```
int ddpi_format_get_ptr_to_mem_xeval_token (
  Ddpi_Info              info,
  Ddpi_Xeval_Context*    context,
  Ddpi_Xeval_Token       obj_token,
  Dwarf_Die              obj_type_die,
  Ddpi_Xeval_Token       mptr_token,
  Dwarf_Die              mptr_type_die,
  Ddpi_Xeval_Token*      ret_this_token,
  Ddpi_Xeval_Token*      ret_mbr_token,
  Ddpi_Error*            error);
```

Parameters

info
Input. This accepts the Ddpi_Info object.

context
Input. If a module map is used, this accepts the Ddpi_Xeval_Context array. The access field in the Ddpi_Xeval_Context object must be set before is passed into any of the ddpi_format operations.

obj_token
Input. This accepts the token with the address of the object.

obj_type_die
Input. This accepts the DW_TAG_structure_type, DW_TAG_union_type, or DW_TAG_class_type DIE.

mptr_token
Input. This accepts the token with the address of the pointer to member.

mptr_type_die
Input. This accepts the DW_TAG_ptr_to_member_type DIE.

ret_this_token
Output. This returns the XEVAL token for the adjusted this pointer.

ret_mbr_token
Output. This returns the XEVAL token for the member.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful retrieval of the XEVAL token for the specified pointer to member.

DW_DLV_NO_ENTRY
Returned if the containing class of `mptr_type_die` is not `obj_type_die` or any of its base classes.

DW_DLV_ERROR
This value is returned if:

- `info` is NULL.
- `context`, `obj_type_die`, `mptr_type_die`, `ret_this_token`, or `ret_mbr_token` is NULL.
- `mptr_type_die` is not a pointer to a member type.
- `DW_AT_use_location` does not exist for the type of `mptr_type_die`.
- `DW_AT_containing_type` does not exist for the type of `mptr_type_die`
- An error occurs.

ddpi_format_set_showbases operation

The `ddpi_format_set_showbases` operation sets whether or not the values of base classes are printed.

Prototype

```
int ddpi_format_set_showbases(
  Dwarf_Bool            show_bases);
```

Parameters

show_bases
Input. This accepts a value for the flag that indicates if base classes printed.

ddpi_format_set_maxstring operation

The `ddpi_format_set_maxstring` operation sets the maximum length of strings printed with the `%<S>` format specifier.

If the value is 0, then there is no maximum.

Prototype

```
int ddpi_format_set_maxstring(
  Dwarf_Unsigned        max_string);
```

Parameters

max_string
Input. This accepts the maximum length of strings.

ddpi_format_set_expand_classes operation

The `ddpi_format_set_expand_classes` operation sets the `%<Nd>` format specifier to expand or not expand classes.

Prototype

```
int ddpi_format_set_expand_classes(
  Dwarf_Bool         expand_classes);
```

Parameters

expand_classes
Input. This accepts a value for the flag that indicates if classes are expanded.

Return values

`ddpi_format_set_expand_classes` always returns `DW_DLV_OK`.

ddpi_format_set_expand_structs operation

The `ddpi_format_set_expand_structs` operation sets the `%<Nd>` format specifiers to expand or not expand structures.

Prototype

```
int ddpi_format_set_expand_structs(
  Dwarf_Bool         expand_structs);
```

Parameters

expand_structs
Input. This accepts a value for the flag that indicates if structures are expanded.

Return values

`ddpi_format_set_expand_structs` always returns `DW_DLV_OK`.

ddpi_format_set_expand_unions operation

The `ddpi_format_set_expand_unions` operation sets the `%<Nd>` format specifier to expand unions or not to expand unions.

Prototype

```
int ddpi_format_set_expand_unions(
  Dwarf_Bool         expand_unions);
```

Parameters

expand_unions
Input. This accepts a value for the flag that indicates if unions are expanded.

Return values

`ddpi_format_set_expand_unions` always returns `DW_DLV_OK`.

ddpi_format_set_expand_enums operation

The `ddpi_format_set_expand_enums` operation sets the `%<Nd>` format specifier to expand or not expand enumerated types.

Prototype

```
int ddpi_format_set_expand_enums(
  Dwarf_Bool         expand_enums);
```

Parameters

expand_enums
Input. This accepts a value for the flag that indicates if enumerated types are expanded.

Return values

`ddpi_format_set_expand_enums` always returns `DW_DLV_OK`.

ddpi_format_set_expand_funcpar operation

The `ddpi_format_set_expand_funcpar` operation sets the `%<Nd>` format specifier to expand or not expand function parameters.

Prototype

```
int ddpi_format_set_expand_funcpar(
  Dwarf_Bool         expand_funcpar);
```

Parameters

expand_funcpar
Input. This accepts a value for the flag that indicates if function parameters are expanded.

Return values

`ddpi_format_set_expand_funcpar` always returns `DW_DLV_OK`.

ddpi_format_set_expand_memfunc operation

The `ddpi_format_set_expand_memfunc` operation sets the `%<Nd>` format specifier to expand or not expand the member functions.

Prototype

```
int ddpi_format_set_expand_memfunc(
  Dwarf_Bool         expand_memfunc);
```

Parameters

expand_memfunc
Input. This accepts a value for the flag that indicates if member functions are expanded.

Return values

`ddpi_format_set_expand_memfunc` always returns `DW_DLV_OK`.

ddpi_format_set_expand_memdata operation

The `ddpi_format_set_expand_memdata` operation sets the `%<Nd>` format specifier to expand or not expand the names of data members.

Prototype

```
int ddpi_format_set_expand_memdata(
  Dwarf_Bool         expand_memdata);
```

Parameters

expand_memdata
Input. This accepts a value for the flag that indicates if data member names are expanded.

Return values

`ddpi_format_set_expand_memdata` always returns `DW_DLV_OK`.

ddpi_format_clear_user_format operation

The `ddpi_clear_user_format` operation clears a user-format specifier for the given type name.

Prototype

```
int ddpi_format_clear_user_format(
  Ddpi_Info             info,
  char*                 typeName,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

typeName
Input. This accepts the type name.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned when the given user-format specifier for the given type name is successfully cleared.

DW_DLV_NO_ENTRY
Returned if a user-format specifier is not found for the type name.

DW_DLV_ERROR
This value is returned when `info` is NULL.

Chapter 31. Ddpi_Xeval APIs

The Ddpi_Xeval APIs are a set of low-level operations. They assist with the evaluation of the DWARF opcodes such as conversions, addition, multiplication, bitwise operations, and string length. Although these APIs are partly an implementation of the DWARF Expression Evaluator, they cannot reside in libdwarf because the operations require system-level information when evaluating the expressions. In order to have access to this information, the expression evaluator must be a part of libddpi.

The Ddpi_Xeval objects contain information about the operations, operands, and results. This information is necessary for the evaluation of the DWARF opcodes. There is also a context object (Ddpi_Xeval_Context) that contains the module, stack state, and machine state information. The Ddpi_Xeval_Context object allows an expression to be evaluated at a particular program, scope, and place in the stack, so that the behavior of a variable, at the time it was called, can be determined.

Note: The operands and results are referred to as tokens in this information.

The callback functions allow a consumer process to use its own implementation to override the default implementation. The callbacks can also extend the default implementation with the opcodes or types of the consumer process. These are used when processing other DWARF expressions. For example, when converting, the consumer might want to use their own conversion rules instead of the standard-C semantics.

The following types are currently not supported by the Ddpi_Xeval APIs:

- DW_ATE_complex_float
- DW_ATE_IBM_complex_float_hex
- DW_ATE_IBM_imaginary_float_hex
- DW_ATE_IBM_packed_decimal
- DW_ATE_IBM_zoned_decimal
- DW_ATE_IBM_register

Ddpi_Xeval_Xtended_Op object

The Ddpi_Xeval_Xtended_Op object contains the DWARF opcode.

Type definition

```
typedef struct Ddpi_Xeval_Xtended_Op {
  Dwarf_Small xx_op;
  Dwarf_Small xx_sub_op;
  Dwarf_Half  xx_pad;
} Ddpi_Xeval_Xtended_Op;
```

Members

xx_op
The mandatory DW_OP_DWARF opcode, defined as type Dwarf_Small.

xx_sub_op
If the opcode is DW_OP_IBM_builtin or DW_OP_IBM_user , this is the sub-opcode. Otherwise, this is 0x00. It is defined as type Dwarf_Small.

© Copyright IBM Corp. 2004, 2009

xx_pad
Currently, this is a reserved member, and must be 0. It is defined as type Dwarf_Half.

Ddpi_Xeval_Token_Kind object

The Ddpi_Xeval_Token_Kind object is a structure that contains information about the type of the token.

Type definition

```
typedef struct Ddpi_Xeval_Token_Kind {
  Dwarf_Small  xk_base_encoded_type;
  Dwarf_Half   xk_size;
  Dwarf_Small  xk_len2;
  Dwarf_Small  xk_len3;
} Ddpi_Xeval_Token_Kind;
```

Members

xk_base_encoded_type
The DW_ATE_-encoded base type, defined as type Dwarf_Small.

xk_size
The physical size of the token in bytes or 0xFFFF to indicate LEB128. Defined as Dwarf_Half.

xk_len2
The member size for complex types, or the number of digits for decimal types. For all other types, it should be 0. Defined as Dwarf_Small.

xk_len3
The number of decimal places for decimal types. For all other types, it should be 0. Defined as Dwarf_Small.

Ddpi_Xeval_Token object

The Ddpi_Xeval_Token object contains information about the token.

Type definition

```
typedef struct Ddpi_Xeval_Token {
   Ddpi_Xeval_Token_Kind xt_kind;
   void *                xt_token;
} Ddpi_Xeval_Token;
```

Members

xt_kind
The type of the token.

xt_token
The token entity.

Ddpi_Xeval_Context object

The Ddpi_Xeval_Context object contains information about a variable at the time and place it was requested from a CDA expression. The variable may be requested from a different program, a different scope, or a different place in the stack. The Ddpi_Xeval_Context object can be saved and used later to retrieve the behavior of the variable at the time it was called.

Type definition

```
typedef struct Ddpi_Xeval_Context_s {
   Dwarf_Small           xeval_context_data_struct_version;
   Ddpi_Module           module;
   Ddpi_StackState       ss;
   Ddpi_MachineState     ms;
   /*-- End of version 0x01 structure ---------------------------------------*/
   Ddpi_Access           access;
   /*-- End of version 0x02 structure ---------------------------------------*/
} Ddpi_Xeval_Context;
```

Members

xeval_context_data_struct_version
The version of the variable.

module
The `Ddpi_Module` object.

Note: The `module` field will be used only if both of the following are true:

- The access field is NULL.
- The `xeval_context_data_struct_version` field is 0x02.

ss The `Ddpi_StackState` object.

ms
The `Ddpi_MachineState` object.

access
The `Ddpi_Access` object.

Ddpi_Xeval_Unary_Func object

The `Ddpi_Xeval_Unary_Func` object contains the prototype of the function used to override the processing of a single unary DWARF op.

It overrides the processing of the op if:

- It has been registered as a unary function for a user type
- It has been registered as an override unary function
- It has been registered as an override conversion function

Type definition

```
typedef int (*Ddpi_Xeval_Unary_Func)(
   Ddpi_Xeval_Token_Kind kind,
   Ddpi_Xeval_Xtended_Op op,
   Ddpi_Xeval_Token      parm,
   Ddpi_Xeval_Token*     result,
   Dwarf_Unsigned*       err_val);
```

Members

kind
Input. This accepts the type of the value that the operation is supposed to return. It is expressed in token terms.

op Input. This accepts the opcode.

parm
Input. This accepts the token, on which the operation will be performed.

result
Output. This returns the result token, which must be non-NULL. Also, the `Ddpi_Xeval_Token` data type must be non-NULL, with enough storage allocated to contain the entity.

err_val
Output. This returns the DDPI_DLE error code.

Ddpi_Xeval_Binary_Func object

The `Ddpi_Xeval_Binary_Func` object contains the prototype for a callback function used to override the processing of a single binary DWARF op.

It overrides the processing of the op if:

- It has been registered as a binary function for a user type
- It has been registered as an override binary function

Type definition

```
typedef int (*Ddpi_Xeval_Binary_Func)(
   Ddpi_Xeval_Token_Kind kind,
   Ddpi_Xeval_Xtended_Op op,
   Ddpi_Xeval_Token      parm1,
   Ddpi_Xeval_Token      parm2,
   Ddpi_Xeval_Token*     result,
Dwarf_Unsigned*       err_val);
```

Members

kind
Input. This accepts the type of the value that the operation is supposed to return. It is expressed in token terms.

op Input. This accepts the opcode.

parm1
Input. This accepts the first token, on which the operation will be performed.

parm2
Input. This accepts the second token, on which the operation will be performed.

result
Output. This returns the result token, which must be non-NULL. Also, the `xt_token` field must be non-NULL with enough storage allocated to contain the entity.

err_val
Output. This returns the DDPI_DLE error code.

ddpi_xeval_eval_unary_op operation

The `ddpi_xeval_eval_unary_op` operation runs the given operation, then compares the given kind parameter with the type of the result.

If the types do not match, then the result of the operation will be converted to the given kind after the operation. The conversion follows standard C conversion rules. The `parm` parameter must have a type that is valid for the operation. If necessary, the consumer can convert the token to the required type with the DW_OP_IBM_conv macro before calling this operation.

Prototype

```
int ddpi_xeval_eval_unary_op(
  Ddpi_Info            info,
  Ddpi_Xeval_Token_Kind kind,
  Ddpi_Xeval_Xtended_Op op,
  Ddpi_Xeval_Token      parm,
  Ddpi_Xeval_Token*     result,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

kind
Input. This accepts the type of the operation, which also determines the type of the result.

op Input. This accepts the `op` code.

parm
Input. This accepts the token, on which the operation will be performed.

result
Output. This returns the result token, which must be non-NULL. Also, the `xt_token` field must be non-NULL with enough storage allocated to contain the entity.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful completion.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is rturned if:

- `info`, `result`, or the associated `xt_token` is NULL
- `kind` is not valid
- The operation isn’t supported or it failed

Invalid data types

For the `DW_OP_abs` and `DW_OP_not` operations, the invalid data types are:

- `DW_ATE_address`
- `DW_ATE_boolean`
- `DW_ATE_float`
- `DW_ATE_IBM_float_hex`

For the `DW_OP_neg` and `DW_OP_IBM_logical_not` operations, the invalid data types are:

- `DW_ATE_address`
- `DW_ATE_boolean`

ddpi_xeval_eval_binary_op operation

The ddpi_xeval_eval_binary_op operation compares the types of the parms, converts them following Standard C conversion rules, and then performs the operation.

The ddpi_xeval_eval_binary_op operation then compares the type of the result to the value of the given kind parameter. If the types do not match, the result of the operation is converted to the given type. The parm parameters must have a type that is valid for the operation. If necessary, the consumer can convert the tokens to the required type with the DW_OP_IBM_conv macro before calling this operation.

Prototype

```
int ddpi_xeval_eval_binary_op(
  Ddpi_Info             info,
  Ddpi_Xeval_Token_Kind kind,
  Ddpi_Xeval_Xtended_Op op,
  Ddpi_Xeval_Token      parm1,
  Ddpi_Xeval_Token      parm2,
  Ddpi_Xeval_Token*     result,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the Ddpi_Info object.

kind
Input. This accepts the type of the operation, which also determines the type of the result.

op Input. This accepts the op code.

parm1
Input. This accepts the first token, on which the operation will be performed.

parm
Input. This accepts the second token, on which the operation will be performed.

result
Output. This returns the result token, which must be non-NULL. Also, the xt_token field must be non-NULL with enough storage allocated to contain the entity.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful completion.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is rturned if:

- info, result, or the associated xt_token is NULL
- kind is not valid
- The operation isn’t supported or it failed

Invalid data types

For the DW_OP_and, DW_OP_or, DW_OP_shl, DW_OP_shr, DW_OP_shra, and DW_OP_xor operations, the invalid data types are:

- DW_ATE_address
- DW_ATE_boolean
- DW_ATE_float
- DW_ATE_IBM_float_hex

For the DW_OP_IBM_builtin, DW_OP_IBM_logical_and, DW_OP_IBM_logical_or, and DW_OP_mod operations, the invalid data types are:

- DW_ATE_address
- DW_ATE_boolean

For the DW_OP_div, DW_OP_mul, DW_OP_minus, DW_OP_plus, DW_OP_le, DW_OP_ge, DW_OP_eq, DW_OP_lt, DW_OP_gt, and DW_OP_ne operations, the invalid data type is DW_ATE_boolean.

ddpi_xeval_override_conv_func operation

The ddpi_xeval_override_conv_func operation overrides the default expression-evaluator behavior for given type conversions.

ddpi_xeval_override_conv_func is registered in the Ddpi_Info object. If the provided override function is NULL, then the type conversion will revert to the default.

Prototype

```
int ddpi_xeval_override_conv_func(
  Ddpi_Info               info,
  Ddpi_Xeval_Token_Kind   from_kind,
  Ddpi_Xeval_Token_Kind   to_kind,
  Ddpi_Xeval_Unary_Func   func,
  Ddpi_Error*             error);
```

Parameters

info
Input. This accepts the Ddpi_Info object.

from_kind
Input. This accepts the type, from which the given function will translate.

to_kind
Input. This accepts the type to which the given function will translate.

func
Input. This accepts the override function that converts the entity from the first given type to the second type.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon a successful call.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is rturned if:

- `info` is NULL
- `from_kind` or `to_kind` are not valid
- An allocation error occurs

ddpi_xeval_override_unary_func operation

The `ddpi_xeval_override_unary_func` operation overrides the default expression-evaluator behavior for a given unary operation.

`ddpi_xeval_override_unary_func` is registered in the `Ddpi_Info` object. If the provided override function is NULL, then the unary operator processing will revert to the default.

Note: It is not possible to override the behavior for a single kind of token without overriding the behavior for all types supported by the op. For example, if you are using the `DW_OP_ADD` op, then you cannot override a float without overriding all types supported by `DW_OP_ADD`.

Prototype

```
int ddpi_xeval_override_unary_func(
  Ddpi_Info               info,
  Ddpi_Xeval_Xtended_Op   op,
  Ddpi_Xeval_Unary_Func   func,
  Ddpi_Error*             error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

op Input. This accepts the `op` code for which this function will be used to override all the default processing.

func
Input. This accepts the override function for the given operation. It must be capable of doing all the required processing for all of the types of tokens that are used.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon a successful call.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `info` is NULL.
- An allocation error occurs.
- The operation is not unary.

ddpi_xeval_override_binary_func operation

The ddpi_xeval_override_binary_func operation overrides the default expression-evaluator behavior for a given binary operation.

ddpi_xeval_override_binary_func is registered in the Ddpi_Info object. If the provided override function is NULL, then the binary operator processing will revert to the default.

Note: It is not possible to override the behavior for a single kind of token without overriding the behavior for all types supported by the op. For example, if you are using the DW_OP_ADD op, then you cannot override a float without overriding all types supported by DW_OP_ADD.

Prototype

```
int ddpi_xeval_override_binary_func(
  Ddpi_Info               info,
  Ddpi_Xeval_Xtended_Op   op,
  Ddpi_Xeval_Binary_Func  func,
  Ddpi_Error*             error);
```

Parameters

info
Input. This accepts the Ddpi_Info object.

op Input. This accepts the op code for which this function will be used to override all the default processing.

func
Input. This accepts the override function for the given operation. It must be capable of doing all the required processing for all of the types of tokens that are used.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon a successful call.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is rturned if:

- info is NULL
- An allocation error occurs
- The operation is not binary

ddpi_xeval_provide_unary_func_user_type_support operation

The ddpi_xeval_provide_unary_func_user_type_support operation registers a processing function to a given user type.

The consumer instance then uses this function to provide the specific support required for a given unary operator based on this user type. For example, if the user type is identified as DW_ATE_IBM_user_type, then a function may be required in order to perform a DW_OP_neg operation.

Prototype

```
int ddpi_xeval_provide_unary_func_user_type_support (
  Ddpi_Info               info,
  Ddpi_Xeval_Xtended_Op   op,
  Ddpi_Xeval_Unary_Func   func,
  Ddpi_Error*             error);
```

Parameters

info
Input. This accepts the Ddpi_Info object.

op Input. This accepts the op code, which defines the user type this function supports.

func
Input. This accepts the function for the given operation, which will associate it with the user type. It must be capable of doing all of the required processing for the given type of token.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon a successful call.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is rturned if:

- info is NULL
- An allocation error occurs
- The operation is not unary

ddpi_xeval_provide_binary_func_user_type_support operation

The ddpi_xeval_provide_binary_func_user_type_support operation registers a processing function to a given user type.

The consumer instance uses this function to provide the specific support required for a given unary operator based on this user type. For example, if the user type is identified as DW_ATE_IBM_user_type, then a function may be required in order to perform a DW_OP_plus operation.

Prototype

```
int ddpi_xeval_provide_binary_func_user_type_support (
  Ddpi_Info               info,
  Ddpi_Xeval_Xtended_Op   op,
  Ddpi_Xeval_Binary_Func  func,
  Ddpi_Error*             error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

op Input. This accepts the `op` code, which defines the user type this function supports.

func
Input. This accepts the function for the given operation, which will associate it with the user type. It must be capable of doing all of the required processing for the given type of token.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon a successful call.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is rturned if:

- `info` is NULL
- An allocation error occurs
- The operation is not binary

ddpi_xeval_engine operation

The `ddpi_xeval_engine` operation performs the operations in the given location-expression list, and then returns the token on top of the stack after the operations are done.

The token entity must be freed by the caller per the following code:

```
ddpi_dealloc(info, ret_val->xt_token, DDPI_DLA_CHUNK)
```

Prototype

```
int ddpi_xeval_engine(
    Ddpi_Info             info,
    Dwarf_Locdesc *       llbuf,
    Dwarf_Half            context_version,
    Ddpi_Xeval_Context *  context,
    Ddpi_Xeval_Token *    initial_token,
    Ddpi_Xeval_Token *    ret_val,
    Ddpi_Error *          error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

llbuf
Input. This accepts the location-expression list.

context_version
Input. This accepts the version of the context array. The constant `DDPI_XEVAL_CONTEXT_VERSION` contains the current version.

context
Input. This accepts the context array.

func
Input. This accepts the function for the given operation, which will associate it with the user type. It must be capable of doing all of the required processing for the given type of token.

initial_token
Input. This accepts a token to initially push onto the stack. It should be NULL if it is not required.

ret_val
Output. This returns the token, which cannot be NULL.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon a successful call.

DW_DLV_NO_ENTRY
Returned if the location-expression list is empty.

DW_DLV_ERROR
This value is rturned if:

- `info` is NULL
- `llbuf` is NULL
- `context` is NULL
- `ret_val` is NULL

Chapter 32. Code set specification APIs

The code set specification operations take ISD information generated by the z/OS XL C/C++ compiler, and create an ELF object file that contains DWARF debugging information. These operations are used by the `isdcnvt` utility. They can also be accessed during run time.

For more information about the utility, see *z/OS Common Debug Architecture User's Guide, SC09-7653*.

Note: For a list of code sets supported by the `iconv` utility, see *z/OS XL C/C++ Programming Guide, SC09-4765*.

The following information describes the operations that allow users to specify the code set that the application uses.

ddpi_info_set_codeset operation

The `ddpi_info_set_codeset` operation specifies the code set for all of the strings (character arrays) that will be passed into `libddpi` operations.

Prototype

```
int ddpi_info_set_codeset(
  Ddpi_Info        info,
  const __ccsid_t  codeset_id,
  __ccsid_t*       prev_cs_id,
  Ddpi_Error*      error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

codeset_id
Input. This accepts the code set for all the strings that will be passed into `libddpi` operations. The user can obtain this ID by calling `__toCcsid()`.

For more information on the `__toCcsid()` function, see the library functions in *z/OS XL C/C++ Run-Time Library Reference*. For a list of code sets that are supported, see *z/OS XL C/C++ Programming Guide*.

prev_cs_id
Output. This returns the code set that was specified in the last call to this operation.

If the operation is called for the first time, this returns ISO8859-1, which is the default code set. If you specify NULL, then the previously specified code set will not be returned.

error
See "The libddpi error parameter" on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the code set information for all of the strings (character arrays) that will be passed into the `libddpi` operations.

© Copyright IBM Corp. 2004, 2009

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- `info` is NULL.
- `codeset_id` is invalid.
- `ddpi_info_set_codeset` is unable to convert the specified code set into an internal code set.

ddpi_format_set_codeset operation

The `ddpi_format_set_codeset` operation specifies the code set for all characters that are stored within the `Ddpi_Space` object.

The formatter requires this information when it formats the characters within the object.

`ddpi_format_set_codeset` performs the same function as `ddpi_format_set_input_charset`.

Note: Although `ddpi_format_set_input_charset` will continue to be supported, it will not be developed further. For this reason, it is recommended that you replace `ddpi_format_set_input_charset` with `ddpi_format_set_codeset`.

Prototype

```
int ddpi_format_set_codeset(
  Ddpi_Info        info,
  const __ccsid_t  codeset_id,
  __ccsid_t*       prev_cs_id,
  Ddpi_Error*      error);
```

Parameters

info
Input. This accepts the `Ddpi_Info` object.

codeset_id
Input. This accepts the code set for all the strings that will be passed into the formatter operations. You can obtain this ID by calling `__toCcsid()`.

For more information on the `__toCcsid()` function, see the library functions in *z/OS XL C/C++ Run-Time Library Reference*. For a list of code sets that are supported, see *z/OS XL C/C++ Programming Guide*.

prev_cs_id
Output. This returns the codeset that was specified in the last call to this operation.

If the operation is called for the first time, this returns IBM-1047, which is the default code set. If you specify NULL, then the previously specified code set will not be returned.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful assignment of the code set format for all of the strings (character arrays) that will be passed into `libddpi` operations.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- An unprintable character is found in the input buffer.
- `info` is NULL.
- `codeset_id` is invalid.
- `ddpi_format_set_codeset` is unable to convert the specified code set into an internal code set.

Chapter 33. Character translation APIs

The ddpi, dwarf, and elf libraries store character data in ASCII format. Values extracted from PPAs or objects may be in EBCDIC. The character translation operations allow the user to quickly translate printable characters between IBM-1047 (EBCDIC) and ISO-8859-1 (ASCII).

ddpi_translate_ibm1047_to_iso8859_1 operation

The ddpi_translate_ibm1047_to_iso8859_1 operation translates character data from IBM-1047 (EBCDIC) to ISO-8859-1 (ASCII).

This operation only translates printable characters. from_buffer is the address of the input string. to_buffer is the address of a section of storage where you should put the output string. A Ddpi_Storagelocn is not used to access either buffer, so the buffers must be in the local address space.

Prototype

```
int ddpi_translate_ibm1047_to_iso8859_1(
  Ddpi_Info             info,
  char*                 to_buffer,
  char*                 from_buffer,
  Dwarf_Unsigned        data_len,
  Dwarf_Unsigned*       ret_actual_len,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the Ddpi_Info object.

to_buffer
Output. This returns the output buffer.

from_buffer
Input. This accepts the input buffer.

data_len
Input. This accepts the number of characters for translation.

ret_actual_len
Output. This returns the number of characters that were translated successfully.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the pointer to the output string storage.

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- An unprintable character is found in the input buffer.
- info is NULL.
- to_buffer, from_buffer, or ret_actual_len pointer is NULL.

© Copyright IBM Corp. 2004, 2009

- data_len is 0.

ddpi_translate_iso8859_1_to_ibm1047 operation

The ddpi_translate_iso8859_1_to_ibm1047 operation translates character data from ISO-8859-1 (ASCII) to IBM-1047 (EBCDIC).

This operation only translates printable characters:

- from_buffer is the address of the input string.
- to_buffer is the address of a section of storage where you can put the output string.

Note: A Ddpi_Storagelocn object is not used to access either buffer, so the buffers must be in the local address space.

Prototype

```
int ddpi_translate_iso8859_1_to_ibm1047(
  Ddpi_Info             info,
  char*                 to_buffer,
  char*                 from_buffer,
  Dwarf_Unsigned        data_len,
  Dwarf_Unsigned*       ret_actual_len,
  Ddpi_Error*           error);
```

Parameters

info
Input. This accepts the Ddpi_Info object.

to_buffer
Output. This returns the output buffer.

from_buffer
Input. This accepts the input buffer that contains the characters to be translated.

data_len
Input. This contains the number of characters in the input buffer.

ret_actual_len
Output. This returns the number of characters that were translated successfully.

error
See “The libddpi error parameter” on page 13.

Return values

DW_DLV_OK
Returned upon successful return of the pointer to the input string storage

DW_DLV_NO_ENTRY
Never returned.

DW_DLV_ERROR
This value is returned if:

- An unprintable character is found in the input buffer.
- info is NULL.
- to_buffer, from_buffer, or ret_actual_len pointer is NULL.
- data_len is 0.

Chapter 34. Code set conversion APIs

ELF objects contain string literals that are encoded in the ISO8859-1 code set. Most CDA operations accept and return string literals encoded in this code set. Applications on the z/OS platform use code sets that are supported by the ICONV function. In fact, the majority of applications on the z/OS platform use string literals encoded in IBM-1047 code set. Before an application encoded in the IBM-1047 code set can use an ELF object, it has to convert the input strings to the ISO8859-1 code set.

For more information about the`iconv` utility, see *z/OS Common Debug Architecture User's Guide, SC09-7653*. For a list of code sets supported by the `iconv` utility, see *z/OS XL C/C++ Programming Guide, SC09-4765*.

The following information describes the operations that allow users to convert input strings to and from ISO8859-1 and IBM-1047 encoding.

ddpi_convert_c_cpp_isdobj operation

The `ddpi_convert_c_cpp_isdobj` operation converts a single GOFF/XOBJ object into the DWARF format.

The object must contain debugging information in the ISD format. The CDA user is responsible for creating an ELF descriptor that will be used to write the converted DWARF debugging information into the ELF object file. `ddpi_convert_c_cpp_isdobj` terminates the descriptor on completion.

Prototype

```
int ddpi_convert_c_cpp_isdobj (
  Ddpi_Space            space,
  Dwarf_Addr            low_addr,
  Dwarf_Addr            high_addr,
  Elf*                  elf,
  int*                  errcode);
```

Parameters

space
Input. This accepts the `Ddpi_Space` object.

low_addr
Input. This accepts the start address in the `Ddpi_Space` object.

high_addr
Input. This accepts the end address in the `Ddpi_Space` object.

elf Input. This accepts the ELF container for the DWARF debugging information.

errcode
Output. This returns `libddpi` error code.

ddpi_fp_convert_c_cpp_isdobj operation

The `ddpi_fp_convert_c_cpp_isdobj` operation converts a single C/C++ object, that contains ISD debugging information, into the DWARF format.

© Copyright IBM Corp. 2004, 2009

ddpi_fp_convert_c_cpp_isdobj will use fopen to open a temporary file for holding the ELF object. When the operation exits, a file pointer will be opened for reading. The operation will also create an ELF object associated with the file pointer.

Prototype

```
int ddpi_fp_convert_c_cpp_isdobj (
  Ddpi_Space            space,
  Dwarf_Addr            start_addr,
  Dwarf_Addr            end_addr,
  FILE**                ret_fp,
  Elf**                 ret_elf,
  int*                  errcode);
```

Parameters

space
Input. This accepts the Ddpi_Space object.

start_addr
Input. This accepts the start address in the Ddpi_Space object.

end_addr
Input. This accepts the end address in the Ddpi_Space object.

ret_fp
Output. This returns a temporary file pointer for the ELF object file.

ret_elf
Output. This returns an ELF descriptor that is used to read from the ELF object file.

errcode
Output. This returns a pointer to the libddpi error code.

Chapter 35. Build information APIs

Build information operations return information about the libddpi build.

ddpi_build_version operation

The ddpi_build_version operation displays the build ID of the ddpi library. Every release/PTF of the ddpi library will have an unique build ID. This information is useful for providing service information to IBM customer support. Calling this function will emit the build ID string (encoded in ISO8859-1) to stdout.

Prototype

```
char*
  ddpi_build_version(void);
```

Return values

Returns the libddpi build ID. The returned string is encoded in ISO8859-1.

Example

```
/* Compile this code with ASCII option */
printf ("Library(ddpi) Level(%s)\n", ddpi_build_version());
```

ddpi_dll_version operation

The ddpi_dll_version operation validates the version of the DLL, and should be used when dynamically linking to the ddpi library. To find the current library version, call the function with '-1' as an argument.

Prototype

```
unsigned int
  ddpi_dll_version(
    unsigned int         ver);
```

Return values

0 DLL version is compatible. The user code is compiled with a libddpi DLL that is the same as the current one, or perhaps earlier.

Any non-zero value.
The returned value is the version of libddpi DLL used for building the user code, which means that the user code is compiled with an libddpi DLL that is more recent than the current library and the DLL version is incompatible.

Example

```
#include
#include "libddpi.h"

dllhandle     *cdadll;
unsigned int (*version_chk)(unsigned int);
unsigned int  dll_version;

#ifdef _LP64
 #define __CDA_DDPI   "CDAEQDPI"
#else
 #define __CDA_DDPI   "CDAEDPI"
#endif
```

© Copyright IBM Corp. 2004, 2009

```
#if LIBDDPI_IS_DLL
 cdadll = dllload(__CDA_DDPI);
 if (cdadll == NULL) {
   /* libddpi DLL not found */
 }

 version_chk = (unsigned int (*)(unsigned int))
            dllqueryfn(cdadll, "ddpi_dll_version");
 if (version_chk == NULL) {
   /* Version API not found, should NEVER happen */
 }

 dll_version = version_chk (LIBDDPI_DLL_VERSION);
 if (dll_version != 0) {
   /* Incompatible DLL version */
 }
#endif
```

Appendix A. libddpi error macros and messages

The following information describes the error macro DDPI_DLE_LAST and the message macros that define the error messages.

DDPI_DLE_LAST error macro

DDPI_DLE_LAST is used to indicate the number of messages that `libddpi` can generate.

Error messages

The table lists the error messages that can be generated by the `libddpi` library.

Note: The message macros that have names that contain ″XFS″ apply to user-defined storage callback functions. XFS stands for data transfer.

Table 4. libddpi error messages

Message macro	Message number	Message
DDPI_DLE_NO_ERROR	0	No error.
DDPI_DLE_INFO_NULL	1	Ddpi_Info object does not exist.
DDPI_DLE_SPACE_NULL	2	Ddpi_Space object does not exist.
DDPI_DLE_PROCESS_NULL	3	Ddpi_Process object does not exist.
DDPI_DLE_THREAD_NULL	4	Ddpi_Thread object does not exist.
DDPI_DLE_MUTEX_NULL	5	Ddpi_Mutex object does not exist.
DDPI_DLE_COND_NULL	6	Ddpi_Cond object does not exist.
DDPI_DLE_LOCK_NULL	7	Ddpi_Lock object does not exist.
DDPI_DLE_POLICY_UNKNOWN	8	The given storagelocn policy is not opaque or transparent.
DDPI_DLE_MODULE_NULL	9	Ddpi_Module object does not exist.
DDPI_DLE_CLASS_NULL	10	Ddpi_Class object does not exist.
DDPI_DLE_SECTION_NULL	11	Ddpi_Section object does not exist.
DDPI_DLE_ENTRYPT_NULL	12	Ddpi_Entrypt object does not exist.
DDPI_DLE_ARANGE_NULL	13	Ddpi_Arange object does not exist.
DDPI_DLE_STORAGELOCN_NULL	14	Ddpi_StorageLocn object does not exist.
DDPI_DLE_STACKSTATE_NULL	15	Ddpi_StackState object does not exist.
DDPI_DLE_MACHINESTATE_NULL	16	Ddpi_Machinestate object does not exist.
DDPI_DLE_STACKSTATE_FN_NULL	17	Ddpi_StackState_Fn object does not exist.

© Copyright IBM Corp. 2004, 2009

Table 4. libddpi error messages (continued)

Message macro	Message number	Message
DDPI_DLE_DATA_CORRUPTION	18	Ddpi detected data corruption of its internal structures.
DDPI_DLE_ACCESS_NULL	19	Ddpi_Access object does not exist.
DDPI_DLE_FILE_NULL	20	Ddpi_File object does not exist.
DDPI_DLE_CONVERT_NULL	21	Ddpi_Convert object does not exist.
DDPI_DLE_IDENTIFY_HANDLER_NULL	22	Ddpi_StackState_Identify handler does not exist.
DDPI_DLE_PARENT_HANDLER_NULL	23	Ddpi_StackState_Parent handler does not exist.
DDPI_DLE_GET_STORAGE_HANDLER_NULL	24	Ddpi_Space handler for ddpi_get_storage() invoked, but does not exist.
DDPI_DLE_SET_STORAGE_HANDLER_NULL	25	Ddpi_Space handler for ddpi_set_storage() invoked, but does not exist.
DDPI_DLE_DEMAND_MODULE_HANDLER_NULL	26	Ddpi_Space handler for ddpi_demand_module() invoked, but does not exist.
DDPI_DLE_DEMAND_ELF_HANDLER_NULL	27	Ddpi_Space handler for ddpi_demand_elf() invoked, but does not exist.
DDPI_DLE_OTHER_HANDLER_NULL	28	Ddpi_Space handler invoked, but does not exist.
DDPI_DLE_NAME_NULL	29	Name is null.
DDPI_DLE_NAME_EMPTY	30	Name is an empty string.
DDPI_DLE_RETURN_PTR_NULL	31	Parameter used to return value is NULL.
DDPI_DLE_ZERO_LENGTH	32	Length must be non-zero.
DDPI_DLE_BUFFER_NULL	33	Data buffer address is NULL.
DDPI_DLE_LIST_DETAIL_NULL	34	List detail element NULL during addition to list.
DDPI_DLE_LIST_EMPTY	35	List should contain elements, but is empty.
DDPI_DLE_ENTRY_NOT_IN_LIST	36	Expected entry not found in list.
DDPI_DLE_ALLOC_INFO_FAIL	37	Allocate Ddpi_Info failed.
DDPI_DLE_ALLOC_SPACE_FAIL	38	Allocate Ddpi_Space failed.
DDPI_DLE_ALLOC_PROCESS_FAIL	39	Allocate Ddpi_Process failed.
DDPI_DLE_ALLOC_THREAD_FAIL	40	Allocate Ddpi_Thread failed.
DDPI_DLE_INVALID_DDPI_ENTITY	41	One of the given Ddpi* entities is not valid.
DDPI_DLE_ALLOC_MODULE_FAIL	42	Allocate Ddpi_Module failed.
DDPI_DLE_ALLOC_CLASS_FAIL	43	Allocate Ddpi_Class failed.
DDPI_DLE_ALLOC_ENTRYPT_FAIL	44	Allocate Ddpi_Entrypt failed.

Table 4. libddpi error messages (continued)

Message macro	Message number	Message
DDPI_DLE_ALLOC_SECTION_FAIL	45	Allocate Ddpi_Section failed.
DDPI_DLE_ALLOC_LOCK_FAIL	46	Allocate Ddpi_Lock failed.
DDPI_DLE_ALLOC_MUTEX_FAIL	47	Allocate Ddpi_Mutex failed.
DDPI_DLE_ALLOC_COND_FAIL	48	Allocate Ddpi_Cond failed.
DDPI_DLE_ALLOC_STORAGELOCN_FAIL	49	Allocate Ddpi_Storagelocn failed.
DDPI_DLE_ALLOC_MACHINESTATE_FAIL	50	Allocate Ddpi_Machinestate failed.
DDPI_DLE_ALLOC_STACKSTATE_FAIL	51	Allocate Ddpi_StackState failed.
DDPI_DLE_ALLOC_STACKSTATE_FN_FAIL	52	Allocate Ddpi_StackState_Fn failed.
DDPI_DLE_ALLOC_SAVEDSTORAGE_FAIL	53	Allocate Ddpi_Savedstorage failed.
DDPI_DLE_ALLOC_SS_DIRECTORY_FAIL	54	Allocate Ddpi_SS_Directory failed.
DDPI_DLE_ALLOC_SSTOR_TOKEN_FAIL	55	Allocate Ddpi_SStor_Token failed.
DDPI_DLE_ALLOC_SS_REPOSITORY_FAIL	56	Allocate Ddpi_SS_Repository failed.
DDPI_DLE_ALLOC_SS_BLOCK_FAIL	57	Allocate Ddpi_SS_Block failed.
DDPI_DLE_ALLOC_STRING_FAIL	58	Allocate string failed.
DDPI_DLE_ALLOC_LIST_FAIL	59	Allocate Ddpi_List failed.
DDPI_DLE_ALLOC_ADDR_FAIL	60	Allocate Ddpi_Addr failed.
DDPI_DLE_ALLOC_CHAIN_FAIL	61	Allocate Ddpi_Chain failed.
DDPI_DLE_ALLOC_LINK_FAIL	62	Allocate Ddpi_Link failed.
DDPI_DLE_ALLOC_OTHERS_FAIL	63	Allocate other storage failed.
DDPI_DLE_INFO_TERM_FAIL	64	Ddpi_Init terminate failed.
DDPI_DLE_SPACE_TERM_FAIL	65	Ddpi_Space terminate failed.
DDPI_DLE_PROCESS_TERM_FAIL	66	Ddpi_Process terminate failed.
DDPI_DLE_THREAD_TERM_FAIL	67	Ddpi_Thread terminate failed.
DDPI_DLE_FLAG_BIT_IDX_BAD	68	The given index was not between 0 and 31 (inclusive).
DDPI_DLE_MODULE_TERM_FAIL	69	Ddpi_Module terminate failed.
DDPI_DLE_CLASS_TERM_FAIL	70	Ddpi_Class terminate failed.
DDPI_DLE_ENTRYPT_TERM_FAIL	71	Ddpi_Entrypt terminate failed.
DDPI_DLE_SECTION_TERM_FAIL	72	Ddpi_Section terminate failed.
DDPI_DLE_LOCK_TERM_FAIL	73	Ddpi_Lock terminate failed.
DDPI_DLE_MUTEX_TERM_FAIL	74	Ddpi_Mutex terminate failed.
DDPI_DLE_COND_TERM_FAIL	75	Ddpi_Cond terminate failed.
DDPI_DLE_CONVERT_TERM_FAIL	76	Ddpi_Convert terminate failed.
DDPI_DLE_STORAGELOCN_TERM_FAIL	77	Ddpi_Storagelocn terminate failed.
DDPI_DLE_MACHINESTATE_TERM_FAIL	78	Ddpi_Machinestate terminate failed.
DDPI_DLE_INFO_MODE_UNKNOWN	79	Unknown Ddpi_Info_Mode value.

Table 4. libddpi error messages (continued)

Message macro	Message number	Message
DDPI_DLE_SPACE_NO_MATCH_MODULE	80	Ddpi_Space does not have a matched Ddpi_Module object.
DDPI_DLE_SPACE_NO_MATCH	81	Ddpi_Space object not matched.
DDPI_DLE_PROCESS_NO_MATCH_THREAD	82	Ddpi_Process does not have a matched Ddpi_Thread object.
DDPI_DLE_PROCESS_NO_MATCH	83	Ddpi_Process object not matched.
DDPI_DLE_MODULE_NO_OWNER	84	Ddpi_Module object has no owner.
DDPI_DLE_MODULE_USAGE_NOT_ZERO	85	Ddpi_Module object's usage is not zero.
DDPI_DLE_MODULE_NO_MATCH_ENTRYPT	86	Ddpi_Module has no matched Ddpi_Entrypt object.
DDPI_DLE_MODULE_NO_MATCH_SPACE	87	Ddpi_Module has no matched Ddpi_Space object.
DDPI_DLE_CLASS_NO_MATCH_SECTION	88	Ddpi_Class has no matched Ddpi_Section object.
DDPI_DLE_CLASS_NO_OWNER	89	Ddpi_Class object has no owner.
DDPI_DLE_CLASS_NO_MATCH	90	Ddpi_Class object is not matched.
DDPI_DLE_CLASS_NOT_IN_ARANGE_TBL	91	Ddpi_Class object is not in the address range table.
DDPI_DLE_CLASS_OWNER_TYPE_UNKNOWN	92	Ddpi_Class object has an unknown owner.
DDPI_DLE_THREAD_NO_MATCH_COND	93	DDpi_Thread has no matched Ddpi_Cond object.
DDPI_DLE_THREAD_NO_MATCH_MUTEX	94	Ddpi_Thread has no matched Ddpi_Mutex object.
DDPI_DLE_THREAD_NO_MATCH_LOCK	95	Ddpi_Thread has no matched Ddpi_Lock object.
DDPI_DLE_THREAD_NO_MATCH	96	Ddpi_Thread object not matched.
DDPI_DLE_MUTEX_NO_MATCH	97	Ddpi_Mutex object not matched.
DDPI_DLE_LOCK_NO_MATCH	98	Ddpi_Lock object not matched.
DDPI_DLE_COND_NO_MATCH	99	Ddpi_Cond object not matched.
DDPI_DLE_SECTION_NO_MATCH	100	Ddpi_Section object not matched.
DDPI_DLE_ENTRYPT_NO_MATCH	101	Ddpi_Entrypt object not matched.
DDPI_DLE_FILE_NO_MATCH	102	Ddpi_File object not matched.
DDPI_DLE_CONVERT_NO_MATCH	103	Ddpi_Convert object not matched.
DDPI_DLE_ACCESS_NO_MATCH	104	Ddpi_Access object not matched.
DDPI_DLE_MACHINESTATE_GPR_COUNT	105	Ddpi_Machinestate GPR count error
DDPI_DLE_MACHINESTATE_FPR_COUNT	106	Ddpi_Machinestate FPR count error
DDPI_DLE_MACHINESTATE_AR_COUNT	107	Ddpi_Machinestate AR count error
DDPI_DLE_MACHINESTATE_CR_COUNT	108	Ddpi_Machinestate CR count error

Table 4. libddpi error messages (continued)

Message macro	Message number	Message
DDPI_DLE_MACHINESTATE_PSW_COUNT	109	Ddpi_Machinestate PSW count error
DDPI_DLE_MACHINESTATE_GPR_VALID_FLAG	110	Ddpi_Machinestate GPR valid flag error
DDPI_DLE_MACHINESTATE_FPR_VALID_FLAG	111	Ddpi_Machinestate FPR valid flag error
DDPI_DLE_MACHINESTATE_CR_VALID_FLAG	112	Ddpi_Machinestate CR valid flag error
DDPI_DLE_MACHINESTATE_AR_VALID_FLAG	113	Ddpi_Machinestate AR valid flag error
DDPI_DLE_MACHINESTATE_PSW_VALID_FLAG	114	Ddpi_Machinestate PSW valid flag error
DDPI_DLE_STORAGE_USERSTORAGE_EXCEED	115	Ddpi_Storage user storage exceeded.
DDPI_DLE_SAVEDSTORAGE_TOKEN_WRONG	116	Ddpi_Savedstorage token error
DDPI_DLE_AMODE_UNKNOWN	117	Addressing mode (AMode) is unknown.
DDPI_DLE_RMODE_UNKNOWN	118	Residency mode (RMode) is unknown.
DDPI_DLE_GS_FAILURE	119	GetStorage routine failure occured.
DDPI_DLE_SS_FAILURE	120	SetStorage routine failure occured.
DDPI_DLE_INSERT_SS_TREE_FAIL	121	Unable to insert entry in Ddpi_SavedStorage tree.
DDPI_DLE_INTERNAL_ERROR	122	libddpi internal error
DDPI_DLE_SAVEDSTORAGE_ALREADY_MAPPED	123	Address range already mapped by another Ddpi_SavedStorage object.
DDPI_DLE_ADDR_EXCEEDS_SPACE_LIMIT	124	New address exceeds maximum address defined by Ddpi_Space object.
DDPI_DLE_XEVAL_OP_NOT_SUPPORTED	125	Opcode not supported for this evaluation/override function.
DDPI_DLE_XEVAL_OP_ERROR	126	Evaluation error
DDPI_DLE_XEVAL_UNKNOWN_TYPE	127	DW_ATE_ type is unknown.
DDPI_DLE_XFS_ADDR_SPACE	128	XFS_addr_space: address space unavailable
DDPI_DLE_XFS_NO_PAGE	129	XFS_no_page: page unavailable
DDPI_DLE_XFS_NO_PAGE_ACCESS	130	XFS_no_page_access: unavailable to access page
DDPI_DLE_XFS_READ_ONLY	131	XFS_no_read_only: memory is read only
DDPI_DLE_XFS_ADDR_BAD	132	XFS_addr_bad: invalid address given

Table 4. libddpi error messages (continued)

Message macro	Message number	Message
DDPI_DLE_XFS_PAGE_BAD	133	XFS_page_bad: invalid page given
DDPI_DLE_XFS_XFER_BAD	134	XFS_xfer_bad: error ocurred during data transfer
DDPI_DLE_ADDR_RANGE_EXCEEDS_SPACE_LIMIT	135	Address range exceeds maximum address defined by Ddpi_Space object.
DDPI_DLE_IP_NOT_FOUND	136	A valid Instruction Pointer was not found.
DDPI_DLE_ADDRESS_ALREADY_MAPPED	137	The address is already assigned to a non-matching class.
DDPI_DLE_DWARF_ERROR	138	A Dwarf_Error occurred.
DDPI_DLE_ALLOC_ELF_FAIL	139	An error occurred while allocating a Ddpi_Elf.
DDPI_DLE_ADDR_NOT_IN_MODULE	140	Given address is not in a known module in this address space.
DDPI_DLE_DDPI_ELF_NULL	141	The Ddpi_Elf is NULL.
DDPI_DLE_EP_NOT_CEESTART	142	The given entry point is not a CEESTART.
DDPI_DLE_IP_NOT_IN_MAPPED_MEMORY	143	The class that contains the Instruction Pointer address has not been defined.
DDPI_DLE_CONTEXT_NULL	144	A null dwarf context version was given to the expression engine.
DDPI_DLE_CONTEXT_VERSION_BAD	145	An invalid dwarf context version was given to the expression engine.
DDPI_DLE_DWARF_LOCDESC_NULL	146	The given dwarf location description was null.
DDPI_DLE_DWARF_LOC_NULL	147	The given dwarf location was null.
DDPI_DLE_ALLOC_ACCESS_FAIL	148	The allocation of a Ddpi_Access failed.
DDPI_DLE_ALLOC_CONVERT_FAIL	149	The allocation of a Ddpi_Convert failed.
DDPI_DLE_ELF_NULL	150	Elf object does not exist.
DDPI_DLE_PPA1_ADDR_NULL	151	Given PPA1 address is null.
DDPI_DLE_PPA2_ADDR_NULL	152	Given PPA2 address is null.
DDPI_DLE_PPA3_ADDR_NULL	153	Given PPA3 address is null.
DDPI_DLE_PPA4_ADDR_NULL	154	Given PPA4 address is null.
DDPI_DLE_PPA_DATA_STRUCT_NULL	155	Given ppa data struct does not exist.
DDPI_DLE_PPA_DATA_UNSUPPORTED_VERSION	156	Given ppa data struct version is not supported.
DDPI_DLE_UNRECOGNIZED_PROLOG	157	The given prolog is not a recognized type.

Table 4. libddpi error messages (continued)

Message macro	Message number	Message
DDPI_DLE_INVALID_CHAR	158	The given character is not a printable character.
DDPI_DLE_ELF_BEGIN_ERROR	159	A call to elf_begin() failed.
DDPI_DLE_ELF_GETIDENT_ERROR	160	A call to elf_getident() failed.
DDPI_DLE_ELF_GETEHDR_ERROR	161	A call to elf32_getehdr() or elf64_getehdr() failed.
DDPI_DLE_ELF_GETSHDR_ERROR	162	A call to elf32_getshdr() or elf64_getshdr() failed.
DDPI_DLE_ELF_STRPTR_ERROR	163	A call to elf_strptr() failed.
DDPI_DLE_ELF_FILE_INVALID	164	ELF file format is invalid.
DDPI_DLE_ELF_FILE_ERROR	165	Error occurred during scan of ELF file.
DDPI_DLE_ELF_SYMBOL_ERROR	166	Error occurred during load of ELF .symtab.
DDPI_DLE_ELF_REL_SYMBOL_ERROR	167	Invalid symbol index encountered in ELF relocation section.
DDPI_DLE_ELF_REL_TYPE_ERROR	168	Invalid relocation type encountered in ELF relocation section.
DDPI_DLE_ELF_REL_SECTION_ERROR	169	Invalid relocation section index encountered in ELF relocation section.
DDPI_DLE_ELF_FILE_TEXT_HAS_BITS	170	ELF .text section sh_type is not SHT_NOBITS.
DDPI_DLE_ELF_PPA2_SYMBOL_MISSING	171	.ppa2 MD5 signature symbol not found in ELF .symtab section.
DDPI_DLE_ALLOC_ELFDETAILS_FAIL	172	Allocate Ddpi_ElfDetails failed.
DDPI_DLE_ALLOC_ELFSYMBOL_FAIL	173	Allocate Ddpi_ElfSymbol failed.
DDPI_DLE_CNVT_METHOD_NULL	174	The convert method was null.
DDPI_DLE_ELFDETAILS_NULL	175	The Ddpi_Elfdetails was null.
DDPI_DLE_TOKEN_SIZE_TOO_SMALL	176	The token size is too small.
DDPI_DLE_RETURN_LOCATION_TYPE_INVALID	177	The return location type is invalid.
DDPI_DLE_NO_SUBRANGE_TAG	178	The array_type tag does not have a subrange_type tag.
DDPI_DLE_SUBRANGE_INVALID	179	The subrange is not contained in the array.
DDPI_DLE_TOKEN_INVALID	180	The xeval token is not valid.
DDPI_DLE_ALLOC_TYPE_UNKNOWN	181	The given storage allocation type is not recognized.
DDPI_DLE_ACCESS_ALREADY_DEFINED	182	The given module already has an access and only one access is allowed per module.
DDPI_DLE_ELF_HAS_NO_DEBUG_DATA	183	The ELF file does not contain any debug information.
DDPI_DLE_DIE_NULL	184	The given DIE is NULL.

Table 4. libddpi error messages (continued)

Message macro	Message number	Message
DDPI_DLE_BAD_COLUMN	185	The given number of columns is invalid.
DDPI_DLE_PTM_DIE_INVALID	186	The DIE is either not a pointer-to-member DIE, or doesn't have the DW_AT_use_location and DW_AT_containing_type attributes.
DDPI_DLE_CODESET_INVALID	187	Invalid codeset specified.
DDPI_DLE_CODESET_CONVERSION_ERROR	188	Error converting between codesets.
DDPI_DLE_STRING_NULL	189	Ddpi_String object is NULL.
DDPI_DLE_INFO_INVALID	190	Ddpi_Info object is not valid.
DDPI_DLE_MUTEX_INVALID	191	Ddpi_Mutex object is not valid.
DDPI_DLE_STORAGELOCN_INVALID	192	Ddpi_StorageLocn object is not valid.
DDPI_DLE_MACHINESTATE_INVALID	193	Ddpi_MachineState object is not valid.
DDPI_DLE_ENTRYPT_INVALID	194	Ddpi_Entrypt object is not valid.
DDPI_DLE_COND_INVALID	195	Ddpi_Cond object is not valid.
DDPI_DLE_SAVEDSTORAGE_INVALID	196	Ddpi_SavedStorage object is not valid.
DDPI_DLE_SECTION_INVALID	197	Ddpi_Section object is not valid.
DDPI_DLE_SPACE_INVALID	198	Ddpi_Space object is not valid.
DDPI_DLE_STACKSTATE_INVALID	199	Ddpi_StackState object is not valid.
DDPI_DLE_MODULE_INVALID	200	Ddpi_Module object is not valid.
DDPI_DLE_LOCK_INVALID	201	Ddpi_Lock object is not valid.
DDPI_DLE_ELF_INVALID	202	Ddpi_Elf object is not valid.
DDPI_DLE_ACCESS_INVALID	203	Ddpi_Access object is not valid.
DDPI_DLE_CLASS_INVALID	204	Ddpi_Class object is not valid.
DDPI_DLE_PROCESS_INVALID	205	Ddpi_Process object is not valid.
DDPI_DLE_THREAD_INVALID	206	Ddpi_Thread object is not valid.
DDPI_DLE_GS_INCOMPLETE	207	The get_storage handler did not get the required number of bytes.
DDPI_DLE_SS_INCOMPLETE	208	The set_storage handler did not set the required number of bytes.
DDPI_DLE_CONVERT_FN_NULL	209	Ddpi_Convert_Fn object does not exist.
DDPI_DLE_CONVERT_HANDLER_NULL	210	Ddpi_Convert_Handler object does not exist
DDPI_DLE_DEBUG_FORMAT_UNSUPPORTED	211	The format of the debugging information is unsupported.

Table 4. libddpi error messages (continued)

Message macro	Message number	Message
DDPI_DLE_ELF_CSECT_OVERLAP	212	The address ranges of the CSECTS in 2 Ddpi_Elf objects partially overlap.
DDPI_DLE_ALLOC_CONVERT_FN_FAIL	213	Allocate Ddpi_Convert_Fn failed.
DDPI_DLE_CONVERTER_INFO_VERSION_BAD	214	Invalid version in the Ddpi_Converter_Info structure.
DDPI_DLE_SYSADATA_FILE_ERROR	215	Error reading the SYSADATA file.
DDPI_DLE_MODULE_READ_ERROR	216	Error reading the module.
DDPI_DLE_CSECT_EXTRACT_ERROR	217	Error extracting the CSECTS from the module.
DDPI_DLE_USER_AREA_INVALID	218	The user area does not contain valid information.
DDPI_DLE_XEVAL_INTDIV	219	Evaluation is terminated by an integer divide by zero exception.
DDPI_DLE_XEVAL_INTOVF	220	Evaluation is terminated by an integer overflow exception.
DDPI_DLE_XEVAL_FLTDIV	221	Evaluation is terminated by a floating point divide by zero exception.
DDPI_DLE_XEVAL_FLTOVF	222	Evaluation is terminated by a floating point overflow exception.
DDPI_DLE_XEVAL_FLTUND	223	Evaluation is terminated by a floating point underflow exception.
DDPI_DLE_XEVAL_FLTRES	224	Evaluation is terminated by a floating point inexact result exception.
DDPI_DLE_XEVAL_FLTINV	225	Evaluation is terminated by an invalid floating point operation exception.
DDPI_DLE_XEVAL_FLTSUB	226	Evaluation is terminated by a subscript is out of range exception.
DDPI_DLE_XEVAL_FLTSIG	227	Evaluation is terminated by a significance exception.
DDPI_DLE_XEVAL_DECDATA	228	Evaluation is terminated by a decimal data exception.
DDPI_DLE_XEVAL_DECDIV	229	Evaluation is terminated by a decimal divide exception.
DDPI_DLE_XEVAL_DECOVF	230	Evaluation is terminated by a decimal overflow exception.
DDPI_DLE_XEVAL_UNKNOWN	231	Evaluation is terminated by an unknown floating point exception.
DDPI_DLE_FUNCTION_NULL	232	The Ddpi_Function is NULL.
DDPI_DLE_VARIABLE_NULL	233	The Ddpi_Variable is NULL.
DDPI_DLE_TYPE_NULL	234	The Ddpi_Type is NULL.
DDPI_DLE_SOURCEFILE_NULL	235	The Ddpi_Sourcefile is NULL.

Appendix B. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

`http:// www.ibm.com/servers/eserver/zseries/zos/bkserv/`

© Copyright IBM Corp. 2004, 2009

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 2004, 2009

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd. Laboratory
B3/KB7/8200/MKM
8200 Warden Avenue
Markham, Ontario L6G 1C7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain services of Common Debug Architecture.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Java™ and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Standards

The `libddpi` library supports the DWARF Version 3 format and ELF application binary interface (ABI).

DWARF was developed by the UNIX International Programming Languages Special Interest Group (SIG). CDA's implementation of DWARF is based on working draft 7 of the DWARF 3 standard.

ELF was developed as part of the System V ABI. It is copyrighted 1997, 2001, The Santa Cruz Operation, Inc. All rights reserved.

Index

A

access policies
  opaque 274
  specifying 274
  transparent 274
access policy
  determining 282
  setting 283
addresses
  validating 23
API types, libddpi
  CDA-application model 3
  conversion 3
  DWARF-expression 3
  support 3
  system-dependent 3
  system-independent 3
APIs
  consumer 2
  producer 2
ASCII
  codeset 7
  compiler option 7

C

CDA
  definition 1
  libraries 2
changes 7
  CDA 7
codeset
  ASCII(ISO8859-1) 7
Common Debug Architecture 1
compiler options
  ASCII 7
  GONUMBER 6
  NOTEST 6
  TEST 6
  XPLINK 7
compiler version requirements 7
consumer
  API 2
conversion
  utility 6

D

data types
  Ddpi_Info_Mode 9
  initialization and termination 9
  libddpi
    initialization and termination 9
Ddpi_finish 11
Ddpi_SavedStorage object
  and opaque access 274
Ddpi_Space_GS_Handler object
  and opaque access 274
  and transparent access 274
Ddpi_Space_SS_Handler object
  and opaque access 274
  and transparent access 274
Ddpi_StorageLocn object
  and access policy 274, 282, 283
  and access to storage 274
  and opaque access 274
DWARF
  converting from ISD format 9
  definition 1
  format 3
  objects 1
DWARF Expression Evaluator 317
Dwarf_Debug 1
Dwarf_P_Debug 1
dwarfdump 7

E

ELF 2
  definition 1
  object file, definition 1
environment
  determining 9
error conditions
  notification of 13
error handling
  ddpi_init parameters 10
error situations
  capturing 13
Executable and Linking Format 2

G

GONUMBER compiler option 6

H

HEAPPOOLS(on) run-time option 7

I

In Store Debug 6
in-store debugging (ISD)
  converting to DWARF format 9
ISD 6
isdcnvt 6

K

keyboard 349

© Copyright IBM Corp. 2004, 2009

L

Language Environment linkages 271
libddpi
  data types
    initialization and termination 9
libddpi library 3
libdwarf library 3
libdwarf objects definition 1
libelf library 2
libraries
  CDA 2
  libddpi 3
  libdwarf 3
  libelf 2

M

memory
  reallocation 10

N

NOTEST compiler option 6

O

object
  consumer 1
  DWARF 1
  ELF object file 1
  libdwarf 1
  producers 1
opaque access
  description 274
operations 11
options
  compiler 6, 7
  run-time 7

P

POSIX threads 63
producer
  API 2
program analysis applications
  examples 10

R

requirements
  CDA 7
  compiler 7
run-time option
  HEAPPOOLS(on) 7

S

sample applications
  dwarfdump 7
shortcut keys 349
standard stack state-handlers
  for Language Environment linkages 271
standards
  DWARF 3
  ELF 2
storage access policy
  specification 274

T

technical support, finding xvi
TEST compiler option 6
threads
  conditional execution 91
  POSIX 63
  variables
    application usage information 91
transparent access
  description 274

U

user area 7
user interface
  accessibility 349
  disability 349
utilities
  dwarfdump 7
  isdcnvt 6

V

variable-length user area 7

X

XPLINK compiler option 7



Program Number: 5694-A01

Printed in USA

SC09-7654-03

