# Table of Contents

## About This Document

## 1 Kernel Reference Pages

- bcm(9F) ................................................................. 24
- bcopy(9F) ............................................................. 25
- biodone(9F) ......................................................... 26
- biowait(9F) .......................................................... 28
- brelse(9F) ........................................................... 29
- buf(9S) ............................................................... 30
- busywait(9F) ......................................................... 34
- bzero(9F) ............................................................ 35
- copyin(9F) .......................................................... 36
- copyout(9F) ........................................................ 37
- csma_alloc(9F) .................................................... 38
- csma_attr_init(9F) ................................................ 40
- csma_attr_setdata(9F) ......................................... 42
- csma_dealloc(9F) ................................................ 44
- csma_decrement(9F) .......................................... 45
- csma_decrement_sig(9F) ...................................... 46
- csma_increment(9F) ............................................ 47
- csma_timeddecrement(9F) ..................................... 48
- csma_timeddecrement_sig(9F) ............................. 50
- csma_tridecrement(9F) ....................................... 52
- csma_value(9F) ................................................... 53
- cv_alloc(9F) ........................................................ 54
- cv_attr_init(9F) ................................................ 55
- cv_attr_setdata(9F) ............................................ 57
- cv_dealloc(9F) .................................................. 59
- cv_broadcast(9F) ................................................ 60
- cv_signal(9F) .................................................... 62
- cv_timedwait(9F) ............................................... 64
- cv_timedwait_sig(9F) ......................................... 66
- cv_wait(9F) ....................................................... 68
- cv_wait_sig(9F) ................................................ 70
- delay, delay_sig(9F) .......................................... 72
- disksort_dequeue(9F) ........................................ 74
- disksort_enqueue(9F) ........................................ 75
- disksort_init_queue(9F) ...................................... 76
- FREE(9F) .......................................................... 77
- getc(9F) ............................................................ 78
- getcb(9F) .......................................................... 79
- getcf(9F) ........................................................... 80
- getblk(9F) ........................................................ 81
- geterror(9F) ....................................................... 82
- get_system_time(9F) ......................................... 83
- gsignal(9F) ...................................................... 84
- iomap_enable_wc(9F) ......................................... 85
- iomap_pagesize(9F) .......................................... 87
- iovec(9S) .......................................................... 88
- kdaemon_proc_create(9F) ................................... 89
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>daemon_proc_exit(9F)</td>
<td>91</td>
</tr>
<tr>
<td>daemon_thread(9F)</td>
<td>92</td>
</tr>
<tr>
<td>daemon_thread_create(9F)</td>
<td>94</td>
</tr>
<tr>
<td>daemon_thread_exit(9F)</td>
<td>96</td>
</tr>
<tr>
<td>daemon_thread_max_thread_proc(9F)</td>
<td>97</td>
</tr>
<tr>
<td>daemon_thread_needs_to_terminate(9F)</td>
<td>99</td>
</tr>
<tr>
<td>daemon_thread_process_pending Suspension(9F)</td>
<td>100</td>
</tr>
<tr>
<td>daemon_thread_wait(9F)</td>
<td>101</td>
</tr>
<tr>
<td>kernel_iomap(9F)</td>
<td>103</td>
</tr>
<tr>
<td>kernel_iomap_public(9F)</td>
<td>106</td>
</tr>
<tr>
<td>kernel_iounmap(9F)</td>
<td>108</td>
</tr>
<tr>
<td>kfree(9F)</td>
<td>109</td>
</tr>
<tr>
<td>kmalloc(9F)</td>
<td>110</td>
</tr>
<tr>
<td>kmem_arena_alloc(9F)</td>
<td>112</td>
</tr>
<tr>
<td>kmem_arena_attr_init(9F)</td>
<td>114</td>
</tr>
<tr>
<td>kmem_arena_create(9F)</td>
<td>117</td>
</tr>
<tr>
<td>kmem_arena_destroy(9F)</td>
<td>119</td>
</tr>
<tr>
<td>kmem_arena_free(9F)</td>
<td>120</td>
</tr>
<tr>
<td>kmem_arena_varalloc(9F)</td>
<td>122</td>
</tr>
<tr>
<td>kr_close_node(9F)</td>
<td>124</td>
</tr>
<tr>
<td>kr_delete_node(9F)</td>
<td>125</td>
</tr>
<tr>
<td>kr_delete_value(9F)</td>
<td>127</td>
</tr>
<tr>
<td>kr_flush(9F)</td>
<td>128</td>
</tr>
<tr>
<td>kr_get_mod_time(9F)</td>
<td>129</td>
</tr>
<tr>
<td>kr_get_node_info(9F)</td>
<td>131</td>
</tr>
<tr>
<td>kr_get_node_names(9F)</td>
<td>133</td>
</tr>
<tr>
<td>kr_get_value(9F)</td>
<td>135</td>
</tr>
<tr>
<td>kr_get_value_names(9F)</td>
<td>137</td>
</tr>
<tr>
<td>kr_get_vinfo(9F)</td>
<td>139</td>
</tr>
<tr>
<td>kr_link_node(9F)</td>
<td>141</td>
</tr>
<tr>
<td>kr_open_node(9F)</td>
<td>143</td>
</tr>
<tr>
<td>kr_release_reference(9F)</td>
<td>145</td>
</tr>
<tr>
<td>kr_set_node_flags(9F)</td>
<td>146</td>
</tr>
<tr>
<td>kr_set_value(9F)</td>
<td>148</td>
</tr>
<tr>
<td>kr_set_value_flags(9F)</td>
<td>150</td>
</tr>
<tr>
<td>kthread_tid_self(9F)</td>
<td>152</td>
</tr>
<tr>
<td>kthreadp_self(9F)</td>
<td>153</td>
</tr>
<tr>
<td>Ktimeout(9F)</td>
<td>154</td>
</tr>
<tr>
<td>ktune_canauto(9F)</td>
<td>156</td>
</tr>
<tr>
<td>ktune_current(9F)</td>
<td>157</td>
</tr>
<tr>
<td>ktune_error(9F)</td>
<td>158</td>
</tr>
<tr>
<td>ktune_event_t(9S)</td>
<td>159</td>
</tr>
<tr>
<td>ktune_get(9F)</td>
<td>160</td>
</tr>
<tr>
<td>ktune_handler(9F)</td>
<td>161</td>
</tr>
<tr>
<td>ktune_id(9F)</td>
<td>164</td>
</tr>
<tr>
<td>ktuneInactive(9F)</td>
<td>165</td>
</tr>
<tr>
<td>ktune_isauto(9F)</td>
<td>166</td>
</tr>
<tr>
<td>ktune_isdefault(9F)</td>
<td>167</td>
</tr>
<tr>
<td>ktune_isdynamic(9F)</td>
<td>168</td>
</tr>
<tr>
<td>ktune_name(9F)</td>
<td>169</td>
</tr>
<tr>
<td>ktune_pending(9F)</td>
<td>170</td>
</tr>
<tr>
<td>ktune_register_handler(9F)</td>
<td>172</td>
</tr>
<tr>
<td>ktune_savedefault(9F)</td>
<td>174</td>
</tr>
<tr>
<td>ktune_simple_constraint(9F)</td>
<td>175</td>
</tr>
<tr>
<td>ktune_simple_constraint_data_t(9S)</td>
<td>178</td>
</tr>
</tbody>
</table>
ktune_simple_dynamic(9F).................................................................179
ktune_unregister_handler(9F)..........................................................181
ktune_validate_powerof2(9F)............................................................183
ktune_validate_zero_or_min(9F).......................................................185
ktune_warning(9F).........................................................................187
ldsid(9F)......................................................................................188
major(9F).....................................................................................189
makedev(9F)..................................................................................190
MALLOC(9F)................................................................................191
map_mem_to_host(9F).....................................................................193
minor(9F).....................................................................................194
minphys(9F)................................................................................195
msg_printfl(9F)............................................................................196
ms_gettimeofday(9F).....................................................................197
mutex_alloc(9F)............................................................................198
mutex_dealloc(9F)........................................................................200
mutex_attr_init(9F)......................................................................201
mutex_attr_setflag(9F)................................................................203
mutex_lock(9F)...........................................................................205
mutex_owned(9F)........................................................................206
mutex_trylock(9F)........................................................................207
mutex_unlock(9F)........................................................................208
p_pgrp(9F)..................................................................................209
panic(9F)....................................................................................210
physio(9F)...................................................................................211
printf(9F)..................................................................................213
priv_policy(9F)...........................................................................214
priv_set(9F)...............................................................................216
privileged_cred(9F)......................................................................218
privlbcopy(9F)............................................................................220
proc_pid_self(9F).........................................................................222
proc_self(9F).............................................................................223
psignal(9F)................................................................................224
putc(9F).....................................................................................225
putcb(9F)...................................................................................226
putc(9F).....................................................................................227
rwlock_alloc(9F).........................................................................228
rwlock_attr_init(9F)...................................................................230
rwlock_attr_setflag(9F)................................................................230
rwlock_dealloc(9F).....................................................................232
rwlock_downgrade(9F)..................................................................234
rwlock_owned(9F).......................................................................235
rwlock_rdlock(9F).......................................................................236
rwlock_tryrdlock(9F)...................................................................237
rwlock_tryupgrade(9F)................................................................238
rwlock_trywrlock(9F)..................................................................239
rwlock_unlock(9F)......................................................................241
rwlock_upgrade(9F).....................................................................242
rwlock_wrlock(9F).......................................................................243
rwlock_wrowned(9F)....................................................................245
rwspin_alloc(9F).........................................................................246
rwspin_attr_init(9F)...................................................................247
rwspin_dealloc(9F)......................................................................249
rwspin_owned(9F).......................................................................250
rwspin_rdlock(9F)......................................................................252
2 WSIO Reference Pages.................................................................297

block_to_raw(9F). ...........................................................................298
class_get_node(9F). ........................................................................299
devsw_entry_t(9S). ..........................................................................300
devsw_table_t(9S). ..........................................................................301
dma_sync_IO(9F). ............................................................................302
driver_addr_probe(9F). ...................................................................304
driver_attach(9E). ..........................................................................306
driver_close(9E). ............................................................................308
driver_dev_init(9E). .........................................................................309
driver_dev_probe(9E). ......................................................................310
driver_if_init(9E). ............................................................................312
driver_install(9E). ..........................................................................313
driver_ioctl(9E). ............................................................................314
driver_ioctl(9F). ............................................................................315
driver_load(9E). .............................................................................316
driverUnload(9E). ...........................................................................318
driver_minor_build(9E). .................................................................320
driver_minphys(9E). ........................................................................321
driver_unload(9E). ..........................................................................318
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver_read(9E)</td>
<td>325</td>
</tr>
<tr>
<td>driver_select(9E)</td>
<td>326</td>
</tr>
<tr>
<td>driver_strategy(9E)</td>
<td>327</td>
</tr>
<tr>
<td>driver_write(9E)</td>
<td>328</td>
</tr>
<tr>
<td>drv_info(9S)</td>
<td>329</td>
</tr>
<tr>
<td>drv_ops(9S)</td>
<td>331</td>
</tr>
<tr>
<td>free_isc(9F)</td>
<td>334</td>
</tr>
<tr>
<td>get_new_isc(9F)</td>
<td>335</td>
</tr>
<tr>
<td>hw_path_t(9S)</td>
<td>336</td>
</tr>
<tr>
<td>install_driver(9F)</td>
<td>337</td>
</tr>
<tr>
<td>io_convert_dev_t(9S)</td>
<td>338</td>
</tr>
<tr>
<td>io_dev_info_t(9S)</td>
<td>339</td>
</tr>
<tr>
<td>io_dev_to_node(9F)</td>
<td>340</td>
</tr>
<tr>
<td>io_dev_to_options(9F)</td>
<td>341</td>
</tr>
<tr>
<td>io_events_t(9S)</td>
<td>342</td>
</tr>
<tr>
<td>io_get_addr(9F)</td>
<td>343</td>
</tr>
<tr>
<td>io_get_devs(9F)</td>
<td>344</td>
</tr>
<tr>
<td>io_get_devsw_len(9F)</td>
<td>346</td>
</tr>
<tr>
<td>io_get_drv_info(9F)</td>
<td>347</td>
</tr>
<tr>
<td>io_get_flags(9F)</td>
<td>348</td>
</tr>
<tr>
<td>io_get_instance(9F)</td>
<td>349</td>
</tr>
<tr>
<td>io_get_mapping_t(9S)</td>
<td>350</td>
</tr>
<tr>
<td>io_get_name(9F)</td>
<td>351</td>
</tr>
<tr>
<td>io_get_node_relation(9F)</td>
<td>352</td>
</tr>
<tr>
<td>io_get_state(9F)</td>
<td>354</td>
</tr>
<tr>
<td>io_get_type(9F)</td>
<td>355</td>
</tr>
<tr>
<td>io_hw_path_to_node(9F)</td>
<td>356</td>
</tr>
<tr>
<td>io_hw_path_to_str(9F)</td>
<td>357</td>
</tr>
<tr>
<td>io_init_hw_path(9F)</td>
<td>358</td>
</tr>
<tr>
<td>io_invoke_devsw(9F)</td>
<td>359</td>
</tr>
<tr>
<td>io_is_legacy_dev(9F)</td>
<td>360</td>
</tr>
<tr>
<td>io_is_legacy_node(9F)</td>
<td>361</td>
</tr>
<tr>
<td>io_mkdev(9F)</td>
<td>362</td>
</tr>
<tr>
<td>io_make_dev_t(9S)</td>
<td>363</td>
</tr>
<tr>
<td>io_mkdev_ext(9F)</td>
<td>364</td>
</tr>
<tr>
<td>io_node_to_hw_path(9F)</td>
<td>365</td>
</tr>
<tr>
<td>io_post_event(9F)</td>
<td>366</td>
</tr>
<tr>
<td>io_post_event_req_t(9S)</td>
<td>367</td>
</tr>
<tr>
<td>io_query(9F)</td>
<td>369</td>
</tr>
<tr>
<td>io_scanall_t(9S)</td>
<td>370</td>
</tr>
<tr>
<td>io_search(9F)</td>
<td>371</td>
</tr>
<tr>
<td>io_str_to_class(9F)</td>
<td>376</td>
</tr>
<tr>
<td>io_str_to_hw_path(9F)</td>
<td>377</td>
</tr>
<tr>
<td>iodone(9F)</td>
<td>378</td>
</tr>
<tr>
<td>iowait(9F)</td>
<td>379</td>
</tr>
<tr>
<td>isc_claim(9F)</td>
<td>380</td>
</tr>
<tr>
<td>isc_table_type(9S)</td>
<td>382</td>
</tr>
<tr>
<td>m_wsio_funcnum(9F)</td>
<td>385</td>
</tr>
<tr>
<td>m_wsio_selcode(9F)</td>
<td>386</td>
</tr>
<tr>
<td>m_wsio_vscc(9F)</td>
<td>387</td>
</tr>
<tr>
<td>mod_wsio_attach_list_add(9F)</td>
<td>388</td>
</tr>
<tr>
<td>mod_wsio_attach_list_remove(9F)</td>
<td>389</td>
</tr>
<tr>
<td>modlink(9S)</td>
<td>390</td>
</tr>
<tr>
<td>modwrapper(9S)</td>
<td>391</td>
</tr>
<tr>
<td>prop_create(9S)</td>
<td>392</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>prop_destroy(9F)</td>
<td>393</td>
</tr>
<tr>
<td>prop_destroy_all(9F)</td>
<td>394</td>
</tr>
<tr>
<td>prop_get(9F)</td>
<td>395</td>
</tr>
<tr>
<td>prop_modify(9F)</td>
<td>397</td>
</tr>
<tr>
<td>prop_size(9F)</td>
<td>398</td>
</tr>
<tr>
<td>raw_to_block(9F)</td>
<td>399</td>
</tr>
<tr>
<td>wsio_activate_probe(9F)</td>
<td>400</td>
</tr>
<tr>
<td>wsio_alloc_mem(9F)</td>
<td>402</td>
</tr>
<tr>
<td>wsio_alloc_mem_handle(9F)</td>
<td>403</td>
</tr>
<tr>
<td>wsio_allocate_dma_handle(9F)</td>
<td>405</td>
</tr>
<tr>
<td>wsio_allocate_shared_mem(9F)</td>
<td>406</td>
</tr>
<tr>
<td>wsio_async_scan(9F)</td>
<td>409</td>
</tr>
<tr>
<td>WSIO_BiG_ENDIAN(9F)</td>
<td>410</td>
</tr>
<tr>
<td>wsio_cfg_inXX(9F)</td>
<td>411</td>
</tr>
<tr>
<td>wsio_cfg_outXX(9F)</td>
<td>412</td>
</tr>
<tr>
<td>wsio_claim_node(9F)</td>
<td>413</td>
</tr>
<tr>
<td>wsio_destroy_legacy(9F)</td>
<td>414</td>
</tr>
<tr>
<td>wsio_dma_pass_thru(9F)</td>
<td>415</td>
</tr>
<tr>
<td>wsio_dma_set_device_attributes(9F)</td>
<td>417</td>
</tr>
<tr>
<td>wsio_drv_data(9S)</td>
<td>420</td>
</tr>
<tr>
<td>wsio_drv_info(9S)</td>
<td>421</td>
</tr>
<tr>
<td>wsio_fastmap_dma_buffer(9F)</td>
<td>422</td>
</tr>
<tr>
<td>wsio_flush_shared_mem(9F)</td>
<td>424</td>
</tr>
<tr>
<td>wsio_free_dma_handle(9F)</td>
<td>426</td>
</tr>
<tr>
<td>wsio_free_mem(9F)</td>
<td>427</td>
</tr>
<tr>
<td>wsio_free_mem_handle(9F)</td>
<td>428</td>
</tr>
<tr>
<td>wsio_free_shared_mem(9F)</td>
<td>429</td>
</tr>
<tr>
<td>wsio_get_active_processor_count(9F)</td>
<td>431</td>
</tr>
<tr>
<td>wsio_get_all_registers(9F)</td>
<td>432</td>
</tr>
<tr>
<td>wsio_get_drv_priv(9F)</td>
<td>433</td>
</tr>
<tr>
<td>wsio_get_ioports(9F)</td>
<td>434</td>
</tr>
<tr>
<td>wsio_get_isc(9F)</td>
<td>436</td>
</tr>
<tr>
<td>wsio_get_legacy(9F)</td>
<td>437</td>
</tr>
<tr>
<td>wsio_get_processor_count(9F)</td>
<td>439</td>
</tr>
<tr>
<td>wsio_get_relationship(9F)</td>
<td>440</td>
</tr>
<tr>
<td>wsio_get_system_params(9F)</td>
<td>441</td>
</tr>
<tr>
<td>wsio_hwpath_to_isc(9F)</td>
<td>442</td>
</tr>
<tr>
<td>wsio_init_map_context(9F)</td>
<td>443</td>
</tr>
<tr>
<td>wsio_install_driver(9F)</td>
<td>444</td>
</tr>
<tr>
<td>wsio_install_drv_event_handler(9F)</td>
<td>446</td>
</tr>
<tr>
<td>wsio_intr_activate(9F)</td>
<td>448</td>
</tr>
<tr>
<td>wsio_intr_alloc(9F)</td>
<td>450</td>
</tr>
<tr>
<td>wsio_intr_assign_cpus(9F)</td>
<td>452</td>
</tr>
<tr>
<td>wsio_intr_deactivate(9F)</td>
<td>454</td>
</tr>
<tr>
<td>wsio_intr_deactivate_nowait(9F)</td>
<td>456</td>
</tr>
<tr>
<td>wsio_intr_free(9F)</td>
<td>458</td>
</tr>
<tr>
<td>wsio_intr_get_assigned_cpu(9F)</td>
<td>460</td>
</tr>
<tr>
<td>wsio_intr_get_attribute(9F)</td>
<td>462</td>
</tr>
<tr>
<td>wsio_intr_get_irq_line(9F)</td>
<td>464</td>
</tr>
<tr>
<td>wsio_intr_get_loc(9F)</td>
<td>466</td>
</tr>
<tr>
<td>wsio_intr_get_txn_info(9F)</td>
<td>468</td>
</tr>
<tr>
<td>wsio_intr_req_loc(9F)</td>
<td>470</td>
</tr>
<tr>
<td>wsio_intr_set ATTRIBUTE(9F)</td>
<td>472</td>
</tr>
<tr>
<td>wsio_intr_set ATTRIBUTE(9F)</td>
<td>474</td>
</tr>
<tr>
<td>wsio_intr_set_irq_line(9F)</td>
<td>476</td>
</tr>
</tbody>
</table>
3 PCI Reference Pages.........................................................................................................................551

CONNECT_INIT_ROUTINE(9F)..................................................................................................................552
PCI_ATTACH_DEV_INIT_ERROR(9F)...........................................................................................................553
cpci_desc_bus_transactions_isc(9F)......................................................................................................554
cpci_get_port_hdl isc(9F)....................................................................................................................556
cpci_read_cfg_uintN_isc(9F)................................................................................................................558
cpci_read_port_uintN_isc(9F)..............................................................................................................560
cpci_unget_port_hdl isc(9F)................................................................................................................562

tables and figures
4 Network Device Driver Reference Pages

DL_DISABMULTI_REQ(9G) ................................................................. 574
DL_ENABMULTI_REQ(9G) ............................................................. 575
DL_GET_STATISTICS_REQ(9G) .................................................... 576
dl_hp_autoneg_sense_t(9S) ....................................................... 577
dl_hp_cmd_info_t(9S) ............................................................... 578
dl_hp_create_info_t(9S) .......................................................... 579
DL_HP_CREATE_VLAN_REQ(9G) ............................................... 585
DL_HP_DELETE_VLAN_REQ(9G) .................................................. 586
DL_HP_MODIFY_VLAN_REQ(9G) .................................................. 587
dl_hp_dest_llc_info_t(9S) ......................................................... 589
dl_hp_drv_event_type_t(9S) ...................................................... 590
dl_hp_drv_features_one_t(9S) ................................................... 592
dl_hp_drv_param_req_type_t(9S) .............................................. 596
dl_hp_duplex_mode_t(9S) ......................................................... 598
dl_hp_encaps_type_t(9S) .......................................................... 599
dl_hp_event_link_cause_t(9S) ................................................... 600
dl_hp_event_type_t(9S) ............................................................ 602
DL_HP_GET_DRV_PARAM_IOCTL(9G) ........................................... 604
DL_HP_GET_MIBSTATS_REQ(9G) ................................................... 606
DL_HP_GET_64BIT_STATS_REQ(9G) ............................................. 607
dl_hp_getinfo_t(9S) ............................................................... 608
dl_hp_get_tcpseg_info_t(9S) ................................................... 610
dl_hp_hdr_length_t(9S) ........................................................... 612
DL_HP_HW_RESET_REQ(9G) ....................................................... 613
dl_hp_hw_state_t(9S) ............................................................. 614
dl_hp_ifadmin_state_t(9S) ....................................................... 615
dl_hp_info_t(9S) ................................................................. 616
ndl_hp_info_type_t(9S) ........................................................... 617
dl_hp_instance_info_t(9S) ....................................................... 618
dl_hp_llc_info_t(9S) .............................................................. 621
dl_hp_mac_type_t(9FS) ............................................................ 622
dl_hp_oop_hdr_t(9S) ............................................................... 623
dl_hp_oop_out_template_t(9S) ................................................. 624
dl_hp_oop_template_t(9S) ........................................................ 625
dl_hp_oop_type_t(9S) .............................................................. 626
dl_hp_op_param_t(9S) ............................................................. 628
dl_hp_op_param_type_t(9S) ..................................................... 629
dl_hp_op_t(9S) ................................................................. 630
dl_hp_pkt_type_t(9S) ............................................................. 631
dl_hp_promisc_state_t(9S) ....................................................... 633
dl_hp_prop_t(9S) ................................................................. 634
DL_HP_RESET_STATS_REQ(9G) .................................................. 642
dl_hp_search_info_t(9S) .......................................................... 643
dl_hp_search_type_t(9S) .......................................................... 644
DL_HP_SET_DRV_PARAM_IOCTL(9G) ............................................ 645
Table of Contents

DL HP SET IFADMIN_REQ(9G).................................................................647
dl_hp_special_params_t(9S).................................................................648
dl_hp_sp_param_type(9S)..................................................................649
dl_hp_sp_params_t(9S)....................................................................650
dl_hp_src_llc_info_t(9S)..................................................................651
dl_hp_tagging_type_t(9S)..................................................................652
DL_HP USAGE_INFO_ACK(9G)..............................................................653
DL_HP USAGE_INFO_REQ(9G)..............................................................654
dl_hp_usage_info_t(9S)....................................................................655
dl_hp_vlan_info_t(9S).......................................................................656
dl_hp_vlan_tag_t(9S).........................................................................657
DL_PHYS_ADDR_REQ(9G)......................................................................658
DL_PROMISCOFF_REQ(9G)..................................................................659
DL_PROMISCON_REQ(9G)....................................................................660
DL_SET_PHYS_ADDR_REQ(9G)..............................................................661
dlp_eventp(9F)...................................................................................662
dlp_inboundp(9F)...............................................................................663
dlp_propp(9F)....................................................................................664
dlp_wakeupp(9F)................................................................................665
driver_build_headerp(9F).................................................................666
driver_controlp(9F)............................................................................667
driver_event_handlerp(9F).................................................................668
driver_outputp(9F).............................................................................669
filter_packet(9F)...............................................................................670
format_link_nice(9F)..........................................................................671
format_link_raw(9F)..........................................................................672
format_link_terse(9F)........................................................................673
get_opt_parms_type(9S)......................................................................674
kget_log_instance(9F)........................................................................675
KLOG CK(9F)......................................................................................676
klogg_write(9F)..................................................................................677
KTRC_CHECK(9F)................................................................................678
KTRC CK(9F).......................................................................................679
ktrc_write(9F)....................................................................................680
netmgr_arg_req_type_t(9S).................................................................681
netmgr_check_class_in_dlpilist(9F)....................................................682
netmgr_check_class_inst_in_dlpilist(9F)............................................683
netmgr_clone_attr(9F).......................................................................684
netmgrclone_attr(9F).........................................................................685
netmgr_clone_nameval(9F).................................................................686
netmgr_conf_delete_attrib(9F).........................................................687
netmgr_conf_delete_idx(9F)...............................................................688
netmgr_conf_delete_nameval_for_idx(9F).........................................689
netmgr_conf_get_atlist_for_idx(9F)..................................................690
netmgr_conf_get_fd(9F).....................................................................691
netmgr_conf_get_idx_list(9F)............................................................692
netmgr_conf_get_indices_for_name(9F)............................................693
netmgr_conf_get_indices_for_nameval(9F)......................................694
netmgr_conf_get_val_for_name_idx(9F)............................................695
netmgr_conf_set_atlist_for_idx(9F)..................................................696
netmgr_conf_set_attrib_value(9F)....................................................697
netmgr_conf_set_idx(9F).................................................................698
netmgr_conf_set_nameval(9F).......................................................699
netmgr_conf_set_opt(9F).................................................................700
netmgr_dlpi_attach(9F)......................................................................701
netmgr_dlpi_detach(9F)......................................................................702
netmgr_dlpi_open(9F)........................................................................703
netmgr_find_num_opts(9F)...............................................................704
netmgr_find_opt(9F).........................................................................705
5 SCSI Reference Pages

dd_aen(9E)..................................................................................................................824
dd_done(9E)..................................................................................................................826
dd_ioctl(9E)..................................................................................................................827
dd_ioctl_okay(9E).........................................................................................................829
dd_io_init(9E)................................................................................................................830
dd_lpt_init(9E)................................................................................................................831
dd_lpt_uninit(9E).........................................................................................................832
dd_lun_init(9E).............................................................................................................833
DD_OPEN_CNT(9G)........................................................................................................834
dd_pass_thru_done(9E).................................................................................................835
dd_pass_thru_okay(9E).................................................................................................836
dd_start(9E)..................................................................................................................837
dd_strategy(9E)..............................................................................................................838
driver_activate_lpt(9F)...............................................................................................840
driver_check_lpt(9F)....................................................................................................842
driver_inspect_lpt_err(9F).........................................................................................844
driver_ioctl(9E)............................................................................................................845
escsi_action(9F).............................................................................................................846
escsi_addr_t(9S).............................................................................................................848
escsi_addr_to_node(9F).................................................................................................849
escsi_aen_t(9S)...............................................................................................................850
ESCSI_APPEND_LIST(9G)..............................................................................................851
escsi_apsw_reg(9F).................................................................852
escsi_async_evt_t(9S)..............................................................855
escsi_bool_t(9S).........................................................................856
escsi_cdbinfo_alloc(9F)..............................................................857
escsi_cdbinfo(9S).........................................................................859
escsi_cdbinfo_flags_t(9S)............................................................860
escsi_cdbinfo_free(9F)...............................................................862
escsi_cdbinfo_init(9F)...............................................................863
escsi_cdbinfo_t(9F).....................................................................866
escsi_cmdx(9F)...........................................................................868
escsi_cmdx_ext(9F).................................................................870
escsi_ctlr_node_cb(9F)...............................................................871
escsi CTLR_reg(9F)..................................................................873
escsi_ctlr_unreg(9F)...............................................................875
escsi_dd_reg(9F).......................................................................877
escsi_ddsw_t(9S)........................................................................878
ESCSI DEQ(9G)..........................................................................881
escsi_deq_active_lpt(9F).............................................................882
ESCSI DEQ_BP(9F)......................................................................884
ESCSI DEQ_BP_FROM_HEAD(9G)................................................885
ESCSI DEQ_BP_FROM_HEAD_FAST(9G).........................................886
ESCSI DEQ_FROM_HEAD(9G).......................................................887
ESCSI DEQ_FROM_HEAD_FAST(9G).............................................888
ESCSI DEQ_FROM_TAIL(9G).......................................................889
ESCSI DEQ_FROM_TAIL_FAST(9G)..............................................890
escsi_dev_t_to_lun(9F)...............................................................891
escsi_dma32_map(9F)..................................................................892
escsi_dma32_unmap(9F).............................................................894
escsi_dma_map(9F)....................................................................895
escsi_dma_parms_t(9S).............................................................897
escsi_dma_req_flags_t(9S).........................................................898
escsi_dma_unmap(9F)................................................................899
escsi_drv_cb(9F).......................................................................901
escsi_enq_active_lpt(9F)............................................................902
ESCSI ENQ_AFTER(9G)............................................................904
ESCSI ENQ_AT_HEAD(9G).........................................................905
ESCSI ENQ_AT_TAIL(9G)..........................................................906
ESCSI ENQ_BEFORE(9G)...........................................................907
ESCSI ENQ_BP_AT_HEAD(9G)....................................................908
ESCSI ENQ_BP_AT_TAIL(9G)......................................................909
escsi_get_addr(9F)....................................................................910
escsi_get_any_path(9F).............................................................912
escsi_get_best_lpt(9F)...............................................................913
escsi_get_class_inst(9F)............................................................914
escsi_get_inq_serial(9F)............................................................915
ESCSI GET_LEG_BUS_ID(9G)....................................................916
ESCSI GET_LEG_LUN_ID(9G)......................................................917
ESCSI GET_LEG_TGT_ID(9G).......................................................918
ESCSI GET_PARENT(9G).............................................................919
escsi_get_options(9F)...............................................................920
escsi_get_path(9F)....................................................................921
escsi_global_t(9S).....................................................................922
ESCSI HOLD(9G).......................................................................923
escsi_if_aen(9F).......................................................................925
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>escsi_if_ctlr_prb_cbfn</td>
<td>927</td>
</tr>
<tr>
<td>escsi_if_flags_t</td>
<td>929</td>
</tr>
<tr>
<td>escsi_if_objt_t</td>
<td>930</td>
</tr>
<tr>
<td>escsi_ifctlr_attr_t</td>
<td>931</td>
</tr>
<tr>
<td>escsi_ifctlr_get</td>
<td>933</td>
</tr>
<tr>
<td>escsi_ifctlr_reg_t</td>
<td>935</td>
</tr>
<tr>
<td>escsi_ifctlr_stat_t</td>
<td>936</td>
</tr>
<tr>
<td>escsi_ifctlr_type_t</td>
<td>937</td>
</tr>
<tr>
<td>escsi_iflpt_get</td>
<td>938</td>
</tr>
<tr>
<td>escsi_iflpt_stat_t</td>
<td>940</td>
</tr>
<tr>
<td>escsi_info_t</td>
<td>941</td>
</tr>
<tr>
<td>ESCSI_Init_Elem</td>
<td>954</td>
</tr>
<tr>
<td>ESCSI_Init_HDR</td>
<td>955</td>
</tr>
<tr>
<td>escsi_init_sema</td>
<td>956</td>
</tr>
<tr>
<td>escsi_iobj_cpu_t</td>
<td>957</td>
</tr>
<tr>
<td>escsi_iobj_get</td>
<td>958</td>
</tr>
<tr>
<td>ESCSI_IOBJ_Lock</td>
<td>959</td>
</tr>
<tr>
<td>escsi_iobj_t</td>
<td>960</td>
</tr>
<tr>
<td>ESCSI_IOBJ_UNLOCK</td>
<td>961</td>
</tr>
<tr>
<td>escsi_ioctl</td>
<td>962</td>
</tr>
<tr>
<td>escsi_iodone</td>
<td>964</td>
</tr>
<tr>
<td>ESCSI_Is_EMPTY</td>
<td>965</td>
</tr>
<tr>
<td>escsi_is_legacy_dev</td>
<td>966</td>
</tr>
<tr>
<td>ESCSI_Is_LPT_ONLINE</td>
<td>967</td>
</tr>
<tr>
<td>ESCSI_Is_MGMT_DEV</td>
<td>968</td>
</tr>
<tr>
<td>ESCSI_Is_NOT_EMPTY</td>
<td>969</td>
</tr>
<tr>
<td>escsi_is_on_q</td>
<td>970</td>
</tr>
<tr>
<td>escsi_kr_close</td>
<td>971</td>
</tr>
<tr>
<td>escsi_kr_delete</td>
<td>972</td>
</tr>
<tr>
<td>escsi_kr_flags_t</td>
<td>973</td>
</tr>
<tr>
<td>escsi_kr_get_settable_attr</td>
<td>974</td>
</tr>
<tr>
<td>escsi_kr_key_t</td>
<td>976</td>
</tr>
<tr>
<td>escsi_kr_lookup_attr</td>
<td>977</td>
</tr>
<tr>
<td>escsi_kr_open</td>
<td>979</td>
</tr>
<tr>
<td>escsi_kr_set_attr</td>
<td>981</td>
</tr>
<tr>
<td>escsi_krid_t</td>
<td>984</td>
</tr>
<tr>
<td>escsi_krid_type_t</td>
<td>985</td>
</tr>
<tr>
<td>escsi_leg_dev_lookup</td>
<td>986</td>
</tr>
<tr>
<td>escsi_leg_lun_close</td>
<td>987</td>
</tr>
<tr>
<td>ESCSI_LEG_LUN_LOCK</td>
<td>989</td>
</tr>
<tr>
<td>escsi_leg_lun_open</td>
<td>990</td>
</tr>
<tr>
<td>escsi_leg_lun_t</td>
<td>992</td>
</tr>
<tr>
<td>ESCSI_LEG_LUN_PSEMA</td>
<td>993</td>
</tr>
<tr>
<td>ESCSI_LEG_LUN_UNLOCK</td>
<td>994</td>
</tr>
<tr>
<td>ESCSI_LEG_LUN_VSEMA</td>
<td>995</td>
</tr>
<tr>
<td>escsi_leg_strategy</td>
<td>996</td>
</tr>
<tr>
<td>escsi_lookup_obj</td>
<td>998</td>
</tr>
<tr>
<td>ESCSI_LPT_LOCK</td>
<td>999</td>
</tr>
<tr>
<td>escsi_lpt_offline</td>
<td>1000</td>
</tr>
<tr>
<td>escsi_lpt_online</td>
<td>1001</td>
</tr>
</tbody>
</table>
A Deprecated and Obsoleted Interfaces

Table of Contents 17
# List of Tables

1. HP-UX 11i Releases
   - Relevant buf Structure
   - iovec Structure Fields
   - uio Structure Fields
2. Device Driver Fields
3. Driver Relevant Structure Fields
4. Driver-Initialized ISC Fields
5. Modlink Structure
6. Modwrapper Structure
7. Values for `type`
8. Values for `assign_type`
9. Values for `capability`
10. Values for `hints`
11. Values for `type`
4. Structure Fields
5. Common Tasks and Task Codes
6. Structure Fields (`ss_N_fmt_flag_type`)
7. Structure Fields (`ss_N_fmt_parms_type`)
5. Ioctls Supported in HP-UX 11i v3
6. Parameter Combinations
5. Parameter Combinations
4. Object, Type, and Address Components
A. Deprecated Interfaces, Their Description, and Replacement
About This Document

This manual provides kernel, WSIO, GIO, PCI, networking, and SCSI reference manpages for HP-UX 11i v3 platforms.

You can find the latest version of this document on the web at:

http://www.hp.com/go/hpux_ddk

Intended Audience

This document is intended for system administrators or developers responsible for porting or writing drivers. Developers are expected to have:

- Experience writing programs in the C language.
- Working knowledge of the basic concepts of writing a driver.
- An understanding of the functionality of the hardware for which the driver is being written.
- Read the HP-UX System Administration Tasks manual and performed system administration.
- Working knowledge of the virtual memory, I/O, and file system areas in the HP-UX and/or UNIX operating systems.

This document is not a tutorial; it is intended for reference.

What's in This Document

The HP-UX 11i v3 Driver Development Reference (DDR) is divided into several chapters, and each contains manpage information:

- Chapter 1 (page 23) contains manpages for the kernel support routines commonly used by I/O drivers.
- Chapter 2 (page 297) contains manpages describing routines and data structures used by drivers to communicate with the WSIO CDIO. It also includes manpages for GIO routines and data structures.
- Chapter 3 (page 551) describes driver functions that are specific to PCI Services.
- Chapter 4 (page 573) contains manual reference pages for data structures, kernel and user space support routines, and macros essential for HP-UX network interface drivers.
- Chapter 5 (page 823) contains SCSI reference pages and a set of commonly used SCSI functions that allow device and interface drivers to be smaller and more supportable.
- Appendix A (page 1107) contains a list of manpages that have been deprecated or obsoleted in HP-UX 11i v3.
- Glossary — A comprehensive list of terms commonly used in the HP-UX 11i v3 Driver Development Reference and HP-UX 11i v3 Driver Development Guide.

Typographical Conventions

This document uses the following conventions.

Audit (5) An HP-UX manpage. In this example, audit is the name and 5 is the section in the HP-UX Reference. On the web and on the Instant Information CD, it may be a hot link to the manpage itself. From the HP-UX command line, you can enter “man audit” or “man 5 audit” to view the manpage. See man(1).

Book Title The title of a book. On the web and on the Instant Information CD, it may be a hot link to the book itself.

KeyCap The name of a keyboard key. Note that Return and Enter both refer to the same key.

Emphasis Text that is emphasized.
Bold Text that is strongly emphasized.

Bold The defined use of an important word or phrase.

ComputerOut Text displayed by the computer.

UserInput Commands and other text that you type.

Command A command name or qualified command phrase.

Variable The name of a variable that you may replace in a command or function or information in a display that represents several possible values.

[] The contents are optional in formats and command descriptions. If the contents are a list separated by |, you must choose one of the items.

{} The contents are required in formats and command descriptions. If the contents are a list separated by |, you must choose one of the items.

... The preceding element may be repeated an arbitrary number of times.

| Separates items in a list of choices.

HP-UX Release Name and Release Identifier

Each HP-UX 11i release has an associated release name and release identifier. The `uname` command with the `-r` option returns the release identifier. This table shows the releases available for HP-UX 11i.

Table 1 HP-UX 11i Releases

<table>
<thead>
<tr>
<th>Release Identifier</th>
<th>Release Name</th>
<th>Supported Processor Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.11.11</td>
<td>HP-UX 11i v1</td>
<td>PA-RISC</td>
</tr>
<tr>
<td>B.11.20</td>
<td>HP-UX 11i v1.5</td>
<td>Intel® Itanium®</td>
</tr>
<tr>
<td>B.11.22</td>
<td>HP-UX 11i v1.6</td>
<td>Intel® Itanium®</td>
</tr>
<tr>
<td>B.11.23</td>
<td>HP-UX 11i v2.0</td>
<td>Intel® Itanium®</td>
</tr>
<tr>
<td>B.11.23</td>
<td>HP-UX 11i v2.0 September 2004</td>
<td>PA RISC and Intel® Itanium®</td>
</tr>
<tr>
<td>B.11.31</td>
<td>HP-UX 11i v3.0 February 2007</td>
<td>PA RISC and Intel® Itanium®</td>
</tr>
</tbody>
</table>

Related Documents

Additional information about the DDR can be found at:
http://www.hp.com/go/hpux_ddk

Other documents in the DDK collection include:

- DDK FAQ
- HP-UX 11i v3 Driver Development Kit Getting Started Guide
- HP-UX 11i v3 Driver Development Guide

HP Encourages Your Comments

HP encourages your comments concerning this document. We are committed to providing documentation that meets your needs.

Send comments to:
feedback@fc.hp.com

Include the document title and manufacturing part number with your comments.
Email and Internet Resources

HPHP program and developer resource materials are available at the following locations:

- HPHP Program E-mail at: hphp.support@hp.com
- HP Hardware Provider program at: http://www.hp.com/go/hphp

Support and Compatibility Disclaimers

Because drivers function at the level of the kernel, HP reminds you of the following:

- Adding your own driver to HP-UX requires relinking the driver into HP-UX. With each new release you must plan on recompiling your driver in order to reinstall it into the new HP-UX kernel. Many header files do not change. However, drivers typically use some header files that could change across releases (you can have some system dependencies).

- HP provides support services for HP products, including HP-UX. Products, including drivers, from non-HP parties receive no support, other than the support of those parts of a driver that rely on the documented behavior of supported HP products.

- If difficulties arise during the development and test phases of writing a driver, HP might provide assistance in isolating problems to determine if:
  - HP hardware is not at fault; and
  - HP software (firmware) is not at fault by removing user-written kernel drivers.

- When HP hardware, software, and firmware are not at fault, you must seek help from the third party from whom you obtained software or hardware.
1 Kernel Reference Pages

This chapter contains reference pages for the kernel support routines commonly used by kernel modules.
NAME

cmp - Compare two byte arrays

SYNOPSIS

#include <sys/kern_svcs.h>

int bcmp(
    void *s1,
    void *s2,
    size_t n
);

PARAMETERS

s1 Pointer to the first byte array.
s2 Pointer to the second byte array.
n Number of bytes to compare.

DESCRIPTION

The bcmp kernel function compares n bytes of the byte arrays starting at s1 and s2. If these n bytes are identical, the function returns zero. If the n bytes are not identical, the function returns the integer value of (s1[k] - s2[k]), where k is the failing byte offset in the array. Unlike strcmp, bcmp does not terminate when it encounters a null byte.

RETURN VALUES

0 The byte arrays are identical.
<0 The byte arrays are different.

CONSTRAINTS

None

SEE ALSO

strcmp(9F)
bcopy(9F)

NAME

bcopy - Copy data from a source buffer to a destination buffer

SYNOPSIS

#include <sys/kern_svcs.h>

void bcopy(
    void *from,
    void *to,
    size_t n
);

PARAMETERS

from Pointer to the source buffer.
to Pointer to the destination buffer.
n Number of bytes to copy.

DESCRIPTION

The bcopy kernel function copies n bytes from a kernel space buffer to another kernel space buffer. The two buffers must not overlap.

To copy data between user space and kernel space, use copyin, copyout, or uiomove.
To copy data between buffers in other user spaces, use privlbcopy.

RETURN VALUES

None

CONSTRAINTS

None

WARNINGS

Do not use the bcopy function for transfers between memory and I/O space. The underlying routines make choices of the optimal transfer code that will probably not be supported on either the I/O bus adapter or the attached interface card. Failure to heed this warning may result in data corruption, an HPMC (High Priority Machine Check), or a call to panic.

SEE ALSO

copyin(9F), copyout(9F), privlbcopy(9F), uiomove(9F)
NAME
biodone – Complete the buffer I/O transaction.

SYNOPSIS
#include <sys/buf.h>
void biodone(
    struct buf *bp
);

PARAMETERS
bp A pointer to a buf structure.

DESCRIPTION
The biodone kernel function completes the buffer I/O transaction. There must be a corresponding call to biowait for the same bp.

If B_CALL is set in bp->b_flags, biodone calls the callback function specified in bp->b_iiodone. The callback function is expected to set the B_DONE flag in bp->b_flags.

If B_CALL is not set in bp->b_flags, biodone marks the buffer I/O as completed by setting the B_DONE flag in bp->b_flags. If B_ASYNC is set, biodone releases the buf structure and associated buffer pointed to by bp, else it resumes the thread waiting on the corresponding call to biowait.

RETURN VALUES
None

CONSTRAINTS
Do not call while holding a spinlock of order >= BUF_HASH_LOCK_ORDER.

WARNINGS
biodone calls panic if B_DONE is set in bp->b_flags upon entry.

EXAMPLE
/*
 * As a sanity check, make sure that B_DONE is not set
 * in b_flags before we call biodone(). If B_DONE is
 * set, then we must be holding a stale buf structure.
 */
VASSERT(!(bp->b_flags & B_DONE));

/*
 * Return I/O completion info in the buf structure.
 */
if (transfer_error) {
    bp->b_error = EIO;
    bp->b_flags |= B_ERROR;
} else {
    bp->b_resid = transfer_residue;
}

/*
 * Complete the buffer I/O transaction. Typically, this
 * results in awaking the thread sleeping in biowait().
 */
biodone(bp);
SEE ALSO

biowait(9F), buf(9F)
NAME
biowait - Wait for the buffer I/O to complete.

SYNOPSIS
#include <sys/buf.h>
int biowait(
    struct buf *bp
);

PARAMETERS
bp Pointer to a buf structure.

DESCRIPTION
The biowait kernel function waits for the completion of the buffer I/O specified by bp. A corresponding call to biodone is required to resume the waiting thread.

RETURN VALUES
0 Successful completion.
<>0 Error.

CONSTRAINTS
Do not call in an interrupt context.
Do not call while holding a spinlock.

EXAMPLES
int error;
struct buf *bp;
...

/*
 * After starting the I/O request, wait for its completion.
 */
error = biowait(bp);

/*
 * biowait returns 0 if the IO completes successfully.
 * A non-zero value is returned if an error has been 
 * encountered, however, the error value returned is not
 * always for the IO completion. To get the IO 
 * completion error that is returned with the buf, we 
 * need to call geterror().
 */
if (error) {
    error = geterror(bp);
}

SEE ALSO
biodone(9F), buf(9F), geterror(9F)
NAME
brese - Release a buffer to the buffer cache.

SYNOPSIS
#include <sys/buf.h>
void brese(  
    struct buf *bp  
);  

PARAMETERS
bp   Pointer to a buf structure.

DESCRIPTION
The brese kernel function releases a buffer to the buffer cache. The buffer header (buf structure)
is pointed to by bp and it must have been previously allocated by a call to geteblk. If there are
treads waiting for this or any free buffer in the buffer cache, the waiting threads are awakened
by brese.

The B_BUSY and B_BCACHE flags must be set in bp->b_flags upon entry to brese.

RETURN VALUES
None

CONSTRAINTS
Do not call while holding a spinlock of order >= BUF_FREE_LOCK_ORDER.

SEE ALSO
buf(9F), geteblk(9F)
NAME
buf – File system buffer header structure for block I/O.

SYNOPSIS
#include <sys/buf.h>

DESCRIPTION
The buf structure describes a file system buffer header used for block I/O. The buffer header contains a pointer to the system buffer allocated to the header and specifies control and status information for the I/O transfer to be performed. Block drivers are passed buffer headers through their driver_strategy routines. Buffer headers and their associated system buffers may be allocated by drivers through geteblk.

The buf structure also describes a buffer header used for raw I/O. The buffer header in this case is formatted by physio and points to a user buffer, not a system buffer. The user buffer is mapped into kernel space for legacy drivers that specify C_MAP_BUFFER_TO_KERNEL in the d_flags field of their drv_opts_t structure.

Do not depend on the size of the buf structure when writing a driver. In particular, do not allocate a buf structure through kmalloc and related memory allocation functions. Doing so may invalidate binary compatibility with future releases of HP-UX.

STRUCTURE MEMBERS
The buf structure is defined in <sys/buf.h>. The following table lists important fields in the buffer and their types.

Table 1-1 Relevant buf Structure

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>av_back</td>
<td>struct buf *</td>
</tr>
<tr>
<td>av_forw</td>
<td>struct buf *</td>
</tr>
<tr>
<td>b_bcount</td>
<td>int32_t</td>
</tr>
<tr>
<td>b_blkno</td>
<td>daddr_t</td>
</tr>
<tr>
<td>b_bufsize</td>
<td>int32_t</td>
</tr>
<tr>
<td>b_dev</td>
<td>dev_t</td>
</tr>
<tr>
<td>b_error</td>
<td>short</td>
</tr>
<tr>
<td>b_flags</td>
<td>int32_t</td>
</tr>
<tr>
<td>b_iqdone</td>
<td>int (<em>)(</em>)()</td>
</tr>
<tr>
<td>b_merge</td>
<td>struct buf *</td>
</tr>
<tr>
<td>b_merge_cnt</td>
<td>uint16_t</td>
</tr>
<tr>
<td>b_resid</td>
<td>unsigned int</td>
</tr>
<tr>
<td>b_s2</td>
<td>intptr_t</td>
</tr>
<tr>
<td>b_s3</td>
<td>char</td>
</tr>
<tr>
<td>b_s7</td>
<td>intptr_t</td>
</tr>
<tr>
<td>b_s8</td>
<td>intptr_t</td>
</tr>
<tr>
<td>b_spaddr</td>
<td>space_t</td>
</tr>
<tr>
<td>Field Name</td>
<td>Type</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td><code>b_un.b_addr</code></td>
<td>caddr_t</td>
</tr>
<tr>
<td><code>b2_flags</code></td>
<td>uint16_t</td>
</tr>
</tbody>
</table>

**av_back, av_forw**

Backward and forward pointers in the buffer headers on the free list where the `B_BUSY` flag is not sent in `b_flags`. Drivers can use the `av_forw` and `av_back` pointers to maintain a queue of busy buffer headers.

**b_bcount**

Number of bytes to be transferred.

**b_blkno**

Block number of the first logical block to be accessed on the target device. A block contains `DEV_BSIZE` bytes.

**b_buFSIZE**

Size of the allocated buffer.

**b_dev**

The `dev_t` major and minor numbers of the target device.

**b_error**

If the `B_ERROR` bit is set in `b_flags`, `b_error` contains the `errno` value for the error that occurred. This field is set by the driver before calling `biodone` and is obtained by calling `geterror`.

**b_flags**

Information about the buffer. This value is composed from the following bit flags:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning if Set</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>B_ASYNC</code></td>
<td>Buffer write is synchronous. Do not wait for I/O completion. Mutually exclusive with <code>B_SYNC</code>.</td>
</tr>
<tr>
<td><code>B_BCACHE</code></td>
<td>The buffer is allocated from the file system buffer cache.</td>
</tr>
<tr>
<td><code>B_BUSY</code></td>
<td>The buffer is in use.</td>
</tr>
<tr>
<td><code>B_CACHE</code></td>
<td><code>bread</code> located this buffer in the cache.</td>
</tr>
<tr>
<td><code>B_CALL</code></td>
<td><code>iodone</code> is to call the function pointed to by <code>b_iiodone</code>.</td>
</tr>
<tr>
<td><code>B_DELWRI</code></td>
<td>Delayed write. Write at exit of <code>avail</code> list processing by the buffer cache management code.</td>
</tr>
<tr>
<td><code>B_DONE</code></td>
<td>The buffer transfer has completed; <code>biodone</code> sets this flag.</td>
</tr>
<tr>
<td><code>B_END_OF_DATA</code></td>
<td>This flag is used to terminate, without error, a <code>physio</code> transfer, with less than <code>b_count</code> bytes transferred.</td>
</tr>
<tr>
<td><code>B_ERROR</code></td>
<td>An error occurred during the I/O transfer. If the driver sets this flag, it must also set the <code>b_error</code> field with an <code>errno</code> value.</td>
</tr>
<tr>
<td><code>B_FSYSIO</code></td>
<td>Buffer came from <code>bread</code> or <code>bwrite</code>.</td>
</tr>
<tr>
<td><code>B_INVAL</code></td>
<td>The buffer does not contain valid information.</td>
</tr>
<tr>
<td><code>B_NDELAY</code></td>
<td>Do not retry on failures.</td>
</tr>
<tr>
<td><code>B_NOCACHE</code></td>
<td>Do not cache data buffer when released.</td>
</tr>
<tr>
<td><code>B_PAGEOUT</code></td>
<td>This flag is used by the buffer cache management system and must not be touched by a driver.</td>
</tr>
<tr>
<td><code>B_PFTIMEOUT</code></td>
<td>With this flag set, a driver is expected to return the I/O request with <code>b_error</code> set to <code>EPOWERF</code> if the device has experienced a power failure. Drivers typically employ a timeout mechanism to detect a device power failure during an I/O transfer.</td>
</tr>
</tbody>
</table>
B_PHYS Indicates the buffer is a user buffer. This flag is normally set by physio.

B_PRIVATE Indicates the buffer header is private to a subsystem such as LVM.

B_RAW Indicates the buffer header is sent to a character (raw) device.

B_READ Data are to be read from the device to host memory. If B_READ is not set, data are to be written from host memory to the device.

B_REWRITE This flag is used by the buffer cache management system and must not be touched by a driver.

B_SYNC Buffer write is synchronous. Wait for I/O completion. Mutually exclusive with B_ASYNC.

B_WANTED One or more threads are sleeping on the buffer header, waiting for the buffer to be freed.

B_WRITE A pseudo flag that semantically indicates not B_READ. The value of B_WRITE is 0; it has no testable bits. To test for a write request, test for the absence of B_READ, as follows:

```c
if (!(bp->b_flags & B_READ) )
```

The expression `(bp->b_flags & B_WRITE)` is always zero.

B_WRITEV This flag is used by LVM when attempting to correct disk soft errors and must not be touched by a driver.

b_iiodone Pointer to a function that iodone calls to complete the I/O request if the B_CALL flag is set in b_flags. The function takes a pointer to the buffer header as its argument and is expected to set the B_DONE flag in b_flags.

b_merge Pointer to the next buf structure where the list of buffers have been logically merged together. Valid only when B2_LIST is set in b2_flags.

b_merge_cnt Number of buffers merged together through b_merge.

b_resid Number of bytes remaining to be transferred; usually set to zero after a successful transfer. The driver_strategy routine sets this field before calling biodone.

b_s2 Scratch field for driver use. For example, SCSI Interface Drivers use this field to store a pointer to a SCSI Control Block.

b_s3 Scratch field for driver use. For example, the SCSI Subsystem uses this field to store state information.

b_s7 Scratch field for driver use. For example, the SCSI Subsystem reserves this field for device drivers.

b_s8 Scratch field for driver use. For example, the SCSI subsystem reserves this field for device drivers.

b_spaddr Space ID of the buffer specified by the buffer header. Do not assume this value to be KERNELSPACE.

b_un.b_addr Virtual address of the buffer specified by the buffer header. The buffer may be mapped in kernel space or it may be mapped in user space. If mapped in user space, b_spaddr must be used with b_addr to form the global virtual address in order to access the buffer. See privlbcopy(9F) for more information.
b2_flags      Information about the buffer in addition to b_flags. This value is composed from the following bit flags:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
<th>if Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2_LIST</td>
<td>Buffer is linked with other buffers through the b_merge field.</td>
<td></td>
</tr>
</tbody>
</table>

SEE ALSO

biodone(9F), biowait(9F), brelse(9F), geteblk(9F), geterror(9F), physio(9F), privlbcopy(9F)
**NAME**

busywait - Wait at least \( t \) microseconds by spinning processor cycles.

**SYNOPSIS**

```
#include <sys/kern_svcsh.h>
void busywait(
    ulong_t t
);
```

**PARAMETERS**

- \( t \) The wait time in microseconds.

**DESCRIPTION**

The `busywait` kernel function waits at least \( t \) microseconds by spinning processor cycles. The processor does no useful work while busy waiting, so keep \( t \) as short as possible.

The `busywait` does not block (that is, sleep), and can be called in an interrupt context or while holding a spinlock. If `busywait` is entered with external interrupts enabled, it is possible for an interrupt to occur and make the actual wait time far exceed the specified \( t \) microseconds.

Use the `delay` kernel function instead if the desired behavior is to block and relinquish the processor.

**RETURN VALUES**

None

**CONSTRAINTS**

None

**EXAMPLES**

```
/*
 * busywait for 5 microseconds
 */

busywait(5UL);
```

**WARNINGS**

On a uniprocessor, it is possible for `busywait` to hang the system since it does not yield the processor if it is called in interrupt context.

**SEE ALSO**

`delay`(9F), `timeout`(9F), `untimeout`(9F).
bzero(9F)

NAME
bzero - Fill a kernel buffer with zeros.

SYNOPSIS
#include <sys/kern_svcs.h>
void bzero(
    void *addr,
    size_t n
);

PARAMETERS
addr Address of kernel buffer.
n Number of bytes to be zeroed.

DESCRIPTION
The bzero kernel function writes \( n \) contiguous bytes of zero, starting at the kernel address specified by \( addr \).

RETURN VALUES
None

CONSTRAINTS
None

SEE ALSO
bcopy(9F)
NAME

copyin - Copy data from a user buffer to a kernel buffer.

SYNOPSIS

#include <sys/kern_svcs.h>

int copyin(
    void *from_user,
    void *to_kernel,
    size_t n
);

PARAMETERS

from_user Source user space address.
to_kernel Destination kernel space address.
n Number of bytes to copy.

DESCRIPTION

The copyin kernel function copies n bytes of data from the user space address from_user to the kernel space address to_kernel. The call to copyin must be made while executing in the user context; that is, while executing in the top half of the driver where the user invokes the driver via a system call such as ioctl.

The copyin function might block (sleep); do not call it while holding a spinlock.

RETURN VALUES

0 Successful completion.
<0 Error.

CONSTRAINTS

Do not call in an interrupt context.
Do not call while holding a spinlock.

WARNINGS

The kernel stack is limited in size. If the buffer to be copied is larger than 128 bytes, avoid allocating buffer space on the kernel stack; instead, allocate the buffer from kernel memory (for example, by calling kmalloc). When large buffers are allocated on the kernel stack, the kernel stack may overflow and cause the kernel to panic.

EXAMPLE

char my_buff[128];
/*
 * Copy from the user buffer to my_buff[] on the kernel stack.
 * Note that buffers larger than 128 bytes must be
 * allocated from kernel memory by calling kmalloc().
 */
if (copyin(user_buf, my_buf, sizeof(my_buf))) {
    return EFAULT;
}

SEE ALSO

bcopy(9F), copyout(9F), privlbcopy(9F), uiomove(9F)
NAME

copyout - Copy data from a kernel buffer to a user buffer.

SYNOPSIS

#include <sys/kern_svcs.h>

int copyout(
    void *from_kernel,
    void *to_user,
    size_t n
);

PARAMETERS

from_kernel Source kernel space address.
to_user Destination user space address.
n Number of bytes to copy.

DESCRIPTION

The copyout kernel function copies n bytes of data from the kernel space address from_kernel to the user space address to_user. The call to copyout must be made while executing in the user context; that is, while executing in the top half of the driver where the user invokes the driver via a system call such as ioctl.

The copyout might block (sleep); do not call it while holding a spinlock.

RETURN VALUES

0 Successful completion.
<>0 Error.

CONSTRAINTS

Do not call in an interrupt context.
Do not call while holding a spinlock.

WARNINGS

The kernel stack is limited in size. If the buffer to be copied is larger than 128 bytes, avoid allocating buffer space on the kernel stack; instead, allocate the buffer from kernel memory (for example, by calling kmalloc). When large buffers are allocated on the kernel stack, the kernel stack may overflow and cause the kernel to panic.

EXAMPLE

char my_buff[128];

/*
 * Copy to the user buffer from my_buff[] on the kernel stack.
 * Note that buffers larger than 128 bytes must be
 * allocated from kernel memory by calling kmalloc().
 */
if (copyout(my_buf, user_buf, sizeof(my_buf))) {
    return EFAULT;
}

SEE ALSO

bcopy(9F), copyin(9F), privlblcopy(9F), uiomove(9F)
NAME
csema_alloc - Allocate and initialize a counting semaphore

SYNOPSIS
#include <sys/csema.h>
csema_t csema_alloc(
    long val,
    char *name,
    csema_attr_t *attr,
    csema_alloc_flg_t alloc_flags
);

PARAMETERS
val The initial value of the semaphore.
name The counting semaphore name.
attr Pointer to the counting semaphore attribute (csema_attr_t) structure or NULL.
alloc_flags CSEMA_WAITOK or CSEMA_NOWAIT; these are mutually exclusive.

DESCRIPTION
The csema_alloc routine allocates and initializes a counting semaphore, setting the initial value to val.
The user can optionally set the maximum count allowed for the semaphore. A debug kernel uses this to verify correctness (see csema_attr_init(9F)).

RETURN VALUES
<>NULL Pointer to the allocated counting semaphore.
NULL The allocation failed because of a lack of memory (in the case of CSEMA_NOWAIT).

CONSTRAINTS
Must be called in a thread context with no spinlocks held, unless CSEMA_NOWAIT is specified.

EXAMPLE
#include <sys/csema.h>
csema_t *csemap;
/*
 * Start with a count of 128 resources.
 */
csemap = csema_alloc(128, "My Counting Semaphore",
    NULL, CSEMA_WAITOK);
/*
 * Code using the counting semaphore.
 */
/*
 * Done with the semaphore.
 */
csema_dealloc(csemap);
SEE ALSO

csema_attr_init(9F), csema_attr_setdata(9F), csema_dealloc(9F), csema_decrement(9F),
csema_decrement_sig(9F), csema_increment(9F), csema_timeddecrement(9F),
csema_timeddecrement_sig(9F), csema_trydecrement(9F), csema_value(9F)
NAME
csema_attr_init - Initialize optional counting semaphore attributes

SYNOPSIS
#include <sys/csema.h>
void csema_attr_init(
    csema_attr_t *attr,
    size_t size
);

PARAMETERS
attr Pointer to the counting semaphore attribute (csema_attr_t) structure.
size The size of the counting semaphore attribute structure.

DESCRIPTION
The csema_attr_init routine initializes the counting semaphore attribute structure. Kernel
modules can pass optional attributes to the lock creation routines through an attribute data
structure. Before using this structure, kernel modules must initialize the structure to its default
values. After the data structure is initialized, the module can set the specific attributes for the
lock with csema_attr_setdata and then call the creation routine.

The csema_attr_init routine is provided for binary compatibility. If new attributes are added
to the attribute structure in the future, you will not have to recompile your code.

Counting semaphores support the following attributes:
CSEMA_ATTR_MAX_CNT The maximum value of the semaphore, if non-zero. Used in debug
kernels to verify correctness.

After the attribute structure is initialized to its default values, use csema_attr_setdata to set
specific values in the attribute structure.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
#include <sys/csema.h>

csema_t *csemap;
csema_attr_t my_attr;
/*
 * Set default values.
 */
csema_attr_init(&my_attr, sizeof(csema_attr_t));
/*
 * Count must not be greater than MAX_TBL_ENTRIES.
 */
csema_attr_setdata(&my_attr, CSEMA_ATTR_MAX_CNT, MAX_TBL_ENTRIES);
/*
 * Start with all entries available - MAX_TBL_ENTRIES.
 */
csemap = csema_alloc(MAX_TBL_ENTRIES, "My Counting Semaphore",
    &my_attr, CSEMA_WAITOK);
SEE ALSO

csema_alloc(9F), csema_attr_init(9F), csema_dealloc(9F), csema_decrement(9F), csema_decrement_sig(9F), csema_increment(9F), csema_timeddecrement(9F), csema_timeddecrement_sig(9F), csema_trydecrement(9F), csema_value(9F)
NAME
csema_attr_setdata - Set specified counting semaphore data in the attribute structure

SYNOPSIS
#include <sys/csema.h>
void csema_attr_setdata(
    csema_attr_t *attr,
    csema_attr_data_t which_attr,
    uintptr_t data
);

PARAMETERS
attr Pointer to the counting semaphore attribute (csema_attr_t) structure.
which_attr Specifies the attribute to set.
data Specifies the value of the attribute to set.

DESCRIPTION
The csema_attr_setdata routine sets specified counting semaphore attribute values in the
counting semaphore attribute structure. Kernel modules can pass optional attributes to the lock
creation routines through an attribute data structure. Before using this structure, kernel modules
must initialize the structure to its default values by using csema_attr_init. After the data
structure is initialized, the module can set the specific attributes for the lock with
csema_attr_setdata and then call the creation routine.

Counting semaphores support the following attributes:
CSEMA_ATTR_MAX_CNT The maximum value of the semaphore, if non-zero. Used in debug
kernels to verify correctness.

RETURN VALUES
None

CONSTRAINTS
Must call csema_attr_init to initialize the attribute structure before calling this routine.

EXAMPLE
#include <sys/csema.h>

csema_t *csemap;
csema_attr_t my_attr;
/*
 * Set default values.
 */
csema_attr_init(&my_attr, sizeof(csema_attr_t));
/*
 * Count must not be greater than MAX_TBL_ENTRIES.
 */
csema_attr_setdata(&my_attr, CSEMA_ATTR_MAX_CNT, MAX_TBL_ENTRIES);
/*
 * Start with all entries available - MAX_TBL_ENTRIES.
 */
csemap = csema_alloc(MAX_TBL_ENTRIES, "My Counting Semaphore",
    &my_attr, CSEMA_WAITOK);
SEE ALSO

csema_alloc(9F), csema_attr_setdata(9F), csema_dealloc(9F), csema_decrement(9F), csema_decrement_sig(9F), csema_decremented(9F), csema_increment(9F), csema_timeddecrement(9F), csema_timeddecrement_sig(9F), csema_trydecrement(9F), csema_value(9F)
csema_dealloc(9F)

NAME
csema_dealloc - Destroy and deallocate a counting semaphore

SYNOPSIS
#include <sys/csema.h>
void csema_dealloc(
    csema_t *csp
);

PARAMETERS
    csp Pointer to the counting semaphore.

DESCRIPTION
The csema_dealloc routine destroys and deallocates a counting semaphore. No threads can
be waiting on the semaphore.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
#include <sys/csema.h>

csema_t *cemap;
/*
 * Start with a count of 128 resources.
 */
cemap = csema_alloc(128, "My Counting Semaphore",
    NULL, CSEMA_WAITOK);

/*
 * Code using the counting semaphore.
 */
/*
 * Done with the semaphore.
 */
csema_dealloc(cemap);

SEE ALSO
csema_alloc(9F), csema_attr_init(9F), csema_attr_setdata(9F), csema_decrement(9F),
csema_decrement_sig(9F), csema_increment(9F), csema_timeddecrement(9F),
csema_timeddecrement_sig(9F), csema_trydecrement(9F), csema_value(9F)
NAME
csema_decrement - Decrement a counting semaphore by a specified value

SYNOPSIS
#include <sys/csema.h>
int csema_decrement(
    csema_t *csp,
    unsigned int val
);

PARAMETERS
csp  Pointer to the counting semaphore.
val  Specifies the amount by which to decrement the counting semaphore.

DESCRIPTION
The csema_decrement routine decrements a counting semaphore by a specified value, blocking
the thread if the value of the count becomes negative. The thread awakens when the count is
incremented by a value sufficient to satisfy this request. If the thread blocks, it cannot be awakened
by a signal.

RETURN VALUES
None

CONSTRAINTS
Must be called from a thread context with no spinlocks held.

EXAMPLE
#include <sys/csema.h>
csema_t *queue_csemap;
/*
 * Wait for X resources.
 */
csema_decrement(queue_csemap, X);
/*
 * Take resources from the queue.
 */
mutex_lock(queue_mtxp);
/*
 * Code to delete X resources from the queue.
 */
mutex_unlock(queue_mtxp);

SEE ALSO
csema_alloc(9F), csema_attr_init(9F), csema_attr_setdata(9F), csema_dealloc(9F),
csema_decrement_sig(9F), csema_increment(9F), csema_timeddecrement(9F),
csema_timeddecrement_sig(9F), csema_trydecrement(9F), csema_value(9F)
csema_decrement_sig(9F)

NAME

csema_decrement_sig - Decrement a counting semaphore by a specified value

SYNOPSIS

#include <sys/csema.h>

int csema_decrement_sig(
    csema_t *csp,
    unsigned int val
);

PARAMETERS

csp Pointer to the counting semaphore.
val Specifies the amount by which to decrement the counting semaphore.

DESCRIPTION

The csema_decrement_sig routine decrements a counting semaphore by a specified value, blocking the thread if the value of the count becomes negative. The thread awakens when the count is incremented by a value sufficient to satisfy this request.

If the wait is interrupted by an HP-UX signal, the thread must inform the caller that an interrupt occurred. Pass this information back up the call stack until the thread returns to user space in order to invoke the application signal handler.

RETURN VALUES

CSEMA_DECR The semaphore was decremented.
CSEMA_INTR The thread was awakened by a signal.

CONSTRAINTS

Must be called from a thread context with no spinlocks held.

EXAMPLE

#include <sys/csema.h>

csema_t *queue_csemap;

if (csema_decrement_sig(queue_csemap, X) == CSEMA_INTR)
    /*
     * Return an error.
     */
else {
    /*
     * Protect the queue.
     */
    mutex_lock(queue_mtxp);
    /*
     * Code to delete X resources from the queue.
     */
    mutex_unlock(queue_mtxp);
}

SEE ALSO

csema_alloc(9F), csema_attr_init(9F), csema_attr_setdata(9F), csema_dealloc(9F), csema_increment(9F),
csema_timeddecrement(9F), csema_timeddecrement_sig(9F), csema_trydecrement(9F), csema_value(9F)
csema_increment(9F)

NAME

csema_increment - Increment a counting semaphore by a specified value

SYNOPSIS

#include <sys/csema.h>

void csema_increment(
    csema_t *csp,
    unsigned int val
);

PARAMETERS

csp Pointer to the counting semaphore.

val Specifies the amount by which to increment the counting semaphore.

DESCRIPTION

The csema_increment routine increments a counting semaphore by a specified value, unblocking any threads whose request can now be granted.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

#include <sys/csema.h>

csema_t *queue_csemap;
/*
 * Return X resources to the queue. Increment counting
 * semaphore by X.
 */
mutex_lock(queue_mtxp);
/*
 * Code to add X resources to the queue.
 */
mutex_unlock(queue_mtxp);
/*
 * Let everyone know about the resources.
 */
csema_increment(queue_csemap, X);

SEE ALSO

csema_alloc(9F), csema_attr_init(9F), csema_attr_setdata(9F), csema_dealloc(9F), csema_decrement(9F),
csema_decrement_sig(9F), csema_timeddecrement(9F), csema_timeddecrement_sig(9F),
csema_trydecrement(9F), csema_value(9F)
NAME

csema_timeddecrement - Decrement a counting semaphore by a specified value

SYNOPSIS

#include <sys/csema.h>

int csema_timeddecrement(
    csema_t *csp,
    unsigned int val,
    timestruct_t *timep
);

PARAMETERS

csp Pointer to the counting semaphore.
val Specifies the amount by which to decrement the counting semaphore.
timep Specifies the length of time to wait, in seconds and nanoseconds.

DESCRIPTION

The csema_timeddecrement routine decrements a counting semaphore by a specified value, blocking the thread if the value of the count becomes negative and returning with an error if a timeout occurs.

If you specify a timer value that requires a finer granularity than the system supports, the timer is rounded up to a supported value.

RETURN VALUES

CSEMA_DECR The semaphore was decremented.
CSEMA_TIMEDOUT The call timed out.

CONSTRAINTS

Must be called from a thread context with no spinlocks held.

EXAMPLE

#include <sys/csema.h>

csema_t *queue_csemap;

if (csema_timeddecrement(queue_csemap, X, &my_timestruct) == CSEMA_TIMEDOUT)
    /*
     * Return an error.
     */
else {
    /*
     * Take resources from the queue. Protect the queue.
     */
    mutex_lock(queue_mtxp);
    /*
     * Code to delete X resources from the queue.
     */
    mutex_unlock(queue_mtxp);
}
SEE ALSO

csema_alloc(9F), csema_attr_init(9F), csema_attr_setdata(9F), csema_dealloc(9F), csema_decrement(9F),
csema_increment(9F), csema_decrement(9F), csema_attribute(9F), csema_decrement(9F), csema_value(9F)
NAME
csema_timeddecrement_sig - Decrement a counting semaphore by a specified value

SYNOPSIS
#include <sys/csema.h>
int csema_timeddecrement_sig(
    csema_t *csp,
    unsigned int val,
    timespec_t *timep
);

PARAMETERS
csp Pointer to the counting semaphore.
val Specifies the amount by which to decrement the counting semaphore.
timep Specifies the length of time to wait, in seconds and nanoseconds.

DESCRIPTION
The csema_timeddecrement_sig routine decrements a counting semaphore by a specified value, blocking the thread if the value of the count becomes negative and returning with an error if a timeout or signal occurs.
If the wait is interrupted by an HP-UX signal, the thread must inform the caller that an interrupt occurred. Pass this information back up the call stack until the thread returns to user space in order to invoke the application signal handler.
If you specify a timer value that requires a finer granularity than the system supports, the timer is rounded up to a supported value.

RETURN VALUES
CSEMA_DECR The semaphore was decremented.
CSEMA_INTR The thread was awakened by a signal.
CSEMA_TIMEDOUT The call timed out.

CONSTRAINTS
Must be called from a thread context with no spinlocks held.

EXAMPLE
#include <sys/csema.h>
csema_t *queue_csemap;
if (csema_timeddecrement_sig(queue_csemap, X, &my_timesruct) != CSEMA_DECR)
    /*
     * Return an error.
     */
else {
    /*
     * Protect the queue.
     */
    mutex_lock(queue_mtxp);
    /*
     * Code to delete X resources from the queue.
     */
mutex_unlock(queue_mtxp);

SEE ALSO

csema_alloc(9F), csema_attr_init(9F), csema_attr_setdata(9F), csema_dealloc(9F), csema_decrement(9F),
csema_decrement_sig(9F), csema_increment(9F), csema_timeddecrement(9F), csema_trydecrement(9F),
csema_value(9F)
NAME

csema_trydecrement - Conditionally decrement a counting semaphore by a specified value

SYNOPSIS

#include <sys/csema.h>

int csema_trydecrement(
    csema_t *csp,
    unsigned int val
);

PARAMETERS

csp  Pointer to the counting semaphore.
val  Specifies the amount by which to increment the counting semaphore.

DESCRIPTION

The csema_trydecrement routine conditionally decrements a counting semaphore by a specified value. If the count becomes negative, the routine returns without decrementing.

RETURN VALUES

<>0  The semaphore was decremented.
  0  The semaphore was not decremented.

CONSTRAINTS

None

EXAMPLE

#include <sys/csema.h>

csema_t *queue_csemap;
if (flags == NOWAIT) {
    if (!csema_trydecrement (queue_csemap, X))
        /*
         * Return an error
         */
    else {
        /* Protect the queue */
        spin_lock (queue_sp);
        /*
         * Code to delete X resources from the queue
         */
        spin_unlock (queue_sp);
    }
}

SEE ALSO

csema_alloc(9F), csema_attr_init(9F), csema_attr_setdata(9F), csema_dealloc(9F), csema_decrement(9F),
csema_decrement_sig(9F), csema_increment(9F), csema_timeddecrement(9F),
csema_timeddecrement_sig(9F), csema_value(9F)
csema_value(9F)

NAME

csema_value - Return the value of a counting semaphore

SYNOPSIS

#include <sys/csema.h>

int csema_value(
    csema_t *csp
);

PARAMETERS

csp Pointer to the counting semaphore.

DESCRIPTION

The csema_value routine returns the current value of a specified counting semaphore.

RETURN VALUES

The current value of the semaphore.

CONSTRAINTS

None

EXAMPLE

#include <sys/csema.h>

csema_t *queue_csemap;

if (csema_value (queue_csemap) < lower_limit)
    /*
     * Create more resources.
     */

SEE ALSO

csema_alloc(9F), csema_attr_init(9F), csema_attr_setdata(9F), csema_dealloc(9F), csema_decrement(9F),
csema_decrement_sig(9F), csema_increment(9F), csema_timeddecrement(9F),
csema_timeddecrement_sig(9F), csema_trydecrement(9F)
NAME

cv_alloc - Allocate and initialize a condition variable

SYNOPSIS

#include <sys/condvar.h>

cv_t cv_alloc(
    char *name,
    cv_attr_t *attr,
    cv_alloc_flag_t alloc_flags
);

PARAMETERS

name The condition variable name.
attr Pointer to the condition variable attribute (cv_attr_t) structure or NULL.
alloc_flags CV_WAITOK or CV_NOWAIT; these are mutually exclusive.

DESCRIPTION

The cv_attr_alloc routine allocates and initializes a condition variable for use with a lock.
The caller can optionally specify the address of the lock to use with the condition variable. A
debug kernel uses this address to verify correctness (see cv_attr_init(9F)).
If CV_WAITOK is specified, this routine might block.

RETURN VALUES

<>NULL Pointer to the allocated condition variable.
NULL The allocation failed because of a lack of memory (in the case of CV_NOWAIT).

CONSTRAINTS

Must be called in a process context with no spinlocks held, if CV_WAITOK is specified.

EXAMPLE

#include <sys/condvar.h>

cv_t *my_cond_var;
my_cond_var = cv_alloc("My Condition Variable", NULL, CV_WAITOK);
/*
 * Code using the cv.
 */
cv_dealloc(my_cond_var);

SEE ALSO

cv_attr_init(9F), cv_attr_setdata(9F), cv_broadcast(9F), cv_dealloc(9F), cv_signal(9F), cv_timedwait(9F),
cv_timedwait_sig(9F), cv_wait(9F), cv_wait_sig(9F)
cv_attr_init(9F)

NAME

cv_attr_init - Initialize optional condition variable attributes

SYNOPSIS

#include <sys/condvar.h>

void cv_attr_init(
    cv_attr_t *attr,
    size_t size
);

PARAMETERS

attr Pointer to the condition variable attribute (cv_attr_t) structure.

size The size of the condition variable attribute structure.

DESCRIPTION

The cv_attr_init routine initializes the condition variable attribute structure. Kernel modules can pass optional attributes to the condition variable creation routines through an attribute data structure. Before using this structure, kernel modules must initialize the structure to its default values. After the data structure is initialized, the module can set the specific attributes for the condition variable with cv_attr_setdata and then call the creation routine.

The cv_attr_init routine is provided for binary compatibility. If new attributes are added to the attribute structure in the future, you will not have to recompile your code.

Condition variables support the following attributes:

CV_ATTR_LOCK The address of the lock to use with this condition variable. The address is used to verify correctness in a debug kernel. The default value is CV_ATTR_ANY_LOCK.

CV_ATTR_FIFO The threads are awakened in strict FIFO (first-in, first-out) order. By default, threads are inserted in the sleep queue based on the greater value of their scheduling priority and PZERO-1.

After the attribute structure is initialized to its default values, use cv_attr_setdata to set specific values in the attribute structure.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

#include <sys/condvar.h>

cv_attr_t my_attr;

cv_t *my_cond_var;

mutex_t my_lock;

/*
 * Set default values.
 */

cv_attr_init(&my_attr, sizeof(cv_attr_t));

/*
 * Only my_lock can be used with the condition variable.
 */
cv_attr_setdata(&my_attr, CV_ATTR_LOCK, &my_lock);

my_cond_var = cv_alloc("My Condition Variable", &my_attr, CV_WAITOK);

SEE ALSO

cv_alloc(9F), cv_attr_setdata(9F), cv_broadcast(9F), cv_dealloc(9F), cv_signal(9F), cv_timedwait(9F),
cv_timedwait_sig(9F), cv_wait(9F), cv_wait_sig(9F)
NAME
cv_attr_setdata - Set specified condition variable attribute values in the attribute structure

SYNOPSIS
#include <sys/condvar.h>
void cv_attr_setdata(
    cv_attr_t *attr,
    cv_attr_data_t which_attr,
    uintptr_t data
);

PARAMETERS
attr Pointer to the condition variable attribute (cv_attr_t) structure.
which_attr Specifies the attribute to set.
data Specifies the value of the attribute to set.

DESCRIPTION
The cv_attr_setdata routine sets specified condition variable attribute values in the condition variable attribute structure. Kernel modules can pass optional attributes to the condition variable creation routines through an attribute data structure. Before using this structure, kernel modules must initialize the structure to its default values. After the data structure is initialized, the module can set the specific attributes for the condition variable with cv_attr_setdata and then call the creation routine.

Condition variables support the following attributes:

CV_ATTR_LOCK The address of the lock to use with this condition variable. The address is used to verify correctness in a debug kernel. The default value is CV_ATTR_ANY_LOCK.

CV_ATTR_FIFO The threads are awakened in strict FIFO (first-in, first-out) order. By default, threads are inserted in the sleep queue based on the greater value of their scheduling priority and PZERO-1.

RETURN VALUES
None

CONSTRAINTS
Must call cv_attr_init to initialize the attribute structure before calling this routine.

EXAMPLE
#include <sys/condvar.h>

cv_attr_t my_attr;
cv_t *my_cond_var;
mutex_t my_lock;
/*
 * Set default values.
 */
cv_attr_init(&my_attr, sizeof(cv_attr_t));
/*
 * Only my_lock can be used with the condition variable.
 */
cv_attr_setdata(&my_attr, CV_ATTR_LOCK, &my_lock);


my_cond_var = cv_alloc("My Condition Variable", &my_attr, CV_WAITOK);

SEE ALSO

cv_alloc(9F), cv_attr_init(9F), cv_broadcast(9F), cv_dealloc(9F), cv_signal(9F), cv_timedwait(9F),
cv_timedwait_sig(9F), cv_wait(9F), cv_wait_sig(9F)
NAME

cv_dealloc - Destroy and deallocate a condition variable

SYNOPSIS

#include <sys/condvar.h>

cv_t cv_dealloc(
    cv_t *cvp
);

PARAMETERS

cvp Pointer to the condition variable to be destroyed.

DESCRIPTION

The cv_dealloc routine destroys and deallocates a dynamically allocated condition variable.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

#include <sys/condvar.h>

cv_t *my_cv;
my_cv = cv_alloc("My Condition Variable", NULL, CV_WAITOK);
/*
 * Code that uses my_cv.
 */
/*
 * Done with my_cv.
 */
cv_dealloc(my_cv);

SEE ALSO

cv_alloc(9F), cv_attr_init(9F), cv_attr_setdata(9F), cv_broadcast(9F), cv_signal(9F), cv_timedwait(9F),
cv_timedwait_sig(9F), cv_wait(9F), cv_wait_sig(9F)
cv_broadcast(9F)

NAME
cv_broadcast - Wake up all threads waiting on a condition

SYNOPSIS
#include <sys/condvar.h>
void cv_broadcast(
    cv_t *cvp,
    void *lock_p,
    cv_lock_type_t lt
);

PARAMETERS
cvp Pointer to the condition variable upon which to wait.
lockp Pointer to the lock protecting the condition variable, if non_NULL.
lt Specifies the type of lock. This can be a spinlock, mutex, reader-writer lock, or
reader-writer spinlock. The caller must specify CV_NULL_LOCK if lockp is NULL.

DESCRIPTION
The cv_broadcast routine wakes up one of the threads that is waiting on the condition
associated with the condition variable. If lockp is non-NULL, the lock is unlocked prior to the
wakeup. After the lock is unlocked, the condition variable will not be accessed by this routine.
If lockp is NULL, the caller must ensure that the condition variable is not destroyed until this
routine returns.

While permissible, do not hold a lock around the cv_broadcast because of the potentially long
path lengths and because the first thing the awakened thread is likely to do is attempt to acquire
the lock.

RETURN VALUES
None

CONSTRAINTS
Spinlocks passed to cv_wait must be less than CV_SPINLOCK_ORDER.

EXAMPLE
#include <sys/condvar.h>

    cv_t *cvp;
    /*
    * Mutex that protects the queue of resources.
    */
    mutex_t *mtxp;

    /* In this subsystem, the cv may be deallocated after ‘waiters’ is zero.
    * Therefore, we might not release the lock prior to the cv_broadcast call and
    * for performance reasons, we do not want to hold the lock around the wakeup.
    * Note, if there were no danger of the cv disappearing, the mutex can be
    * released after waiters was zeroed. In which case, NULL is passed
    * to cv_broadcast.
    */
    mutex_lock (mtxp);
    if (waiters) {
        waiters = 0;
        cv_broadcast(cv, mtxp, CV_MUTEX);
    } else mutex_unlock(mtxp);
SEE ALSO

cv_alloc(9F), cv_attr_init(9F), cv_attr_setdata(9F), cv_dealloc(9F), cv_signal(9F), cv_timedwait(9F),
cv_timedwait_sig(9F), cv_wait(9F), cv_wait_sig(9F)
**NAME**

`cv_signal` - Wake up a thread waiting on a condition

**SYNOPSIS**

```c
#include <sys/condvar.h>

void cv_signal(
    cv_t *cvp,
    void *lock_p,
    cv_lock_type_t lt
);
```

**PARAMETERS**

- `cvp` Pointer to the condition variable upon which to wait.
- `lockp` Pointer to the lock protecting the condition variable, if non_NULL.
- `lt` Specifies the type of lock. This can be a spinlock, mutex, reader-writer lock, or reader-writer spinlock. The caller must specify `CV_NULL_LOCK` if `lockp` is NULL.

**DESCRIPTION**

The `cv_signal` routine wakes up one of the threads that is waiting on the condition associated with the condition variable. If `lockp` is non-NULL, the lock is unlocked prior to the wakeup. After the lock is unlocked, the condition variable will not be accessed by this routine. If `lockp` is NULL, the caller must ensure that the condition variable is not destroyed until this routine returns.

While permissible, do not hold a lock around the `cv_signal` because of the potentially long path lengths and because the first thing the awakened thread is likely to do is attempt to acquire the lock.

**RETURN VALUES**

None

**CONSTRAINTS**

Spinlocks passed to `cv_wait` must be less than `CV_SPINLOCK_ORDER`.

**EXAMPLE**

```c
#include <sys/condvar.h>

cv_t *cvp;

/*
 * Mutex that protects the queue of resources.
 */
mutex_t *mtxp;

/* In this subsystem, the cv might be deallocated after 'waiters' is zero.
 * Therefore, we might not release the lock prior to the cv_signal call and
 * for performance reasons, we do not want to hold the lock around the wakeup.
 * Note, if there were no danger of the cv disappearing, the mutex might be
 * released after waiters was decremented. In which case, NULL is passed
 * to cv_signal.
 */
mutex_lock (mtxp);
if (waiters) {
    waiters = waiters -1;
    cv_signal(cv, mtxp, CV_MUTEX);
} else mutex_unlock(mtxp);
```
SEE ALSO

`cv_alloc(9F), cv_attr_init(9F), cv_attr_setdata(9F), cv_broadcast(9F), cv_dealloc(9F), cv_timedwait(9F),
cv_timedwait_sig(9F), cv_wait(9F), cv_wait_sig(9F)`
NAME

cv_timedwait - Wait for the condition associated with a condition variable or a timeout

SYNOPSIS

#include <sys/condvar.h>

int cv_timedwait(
    cv_t *cvp,
    void *lock_p,
    cv_lock_type_t lt,
    cvw_flags_t flags,
    timestruc_t *timep
    )
;

PARAMETERS

cvp      Pointer to the condition variable upon which to wait.
lockp    Pointer to the lock protecting the condition variable.
lt       Specifies the type of lock. This can be a spinlock, mutex, reader-writer lock, or
          reader-writer spinlock.
flags    Specifies the options or modifiers to the function. Must be the logical OR of one or more
          of the flags defined by the cvw_flags_t enumerated data type. See the DESCRIPTION
          section for more information.
timep    The length of time, in seconds and nanoseconds, to wait.

DESCRIPTION

The cv_timedwait routine waits for either the occurrence of a condition associated with a
condition variable or a timeout. It atomically blocks the calling thread and unlocks the lock; this
guarantees that the thread will not miss the signaling of a condition change. The cv_timedwait
routine returns to the thread with the lock locked, unless CV_NO_RELOCK is specified.
The thread must be able to handle spurious wakeups. There is no guarantee that the condition
is true when the thread resumes execution.

If a timer value is specified that requires finer granularity than the system supports, the timer is
rounded up to a supported value.

The cv_lock_type_t enumerated data type defines the following lock types:
CV_MUTEX      The lock is an adaptive mutex.
CV_REC_MUTEX  The lock is a recursive mutex lock.
CV_RWLOCK_RD  The lock is a read lock.
CV_RWLOCK_WR  The lock is a write lock.
CV_RWSPIN_RD  The lock is a read spinlock. You can only lock the lock one time.
CV_RWSPIN_WR  The lock is a write spinlock. You can only lock the lock one time.
CV_SPIN       The lock is a spinlock.

The cvw_flags_t enumerated data type defines the following flags:
CV_DFLT_FLG   The default value for the flag. Use this value if no other options are specified.
CV_NO_RELOCK  Do not relock the lock before returning to the user. After the lock is unlocked,
              the cv_wait routines will not access the condition variable.

Threads are inserted in the sleep queue based on their scheduling priority or NZERO-1, whichever
is greater, unless the condition variable was created with the CV_ATTR_FIFO option.
RETURN VALUES

CV_SIGNALED  The condition was met and the function returned because of a call to 
\texttt{cv_signal} or \texttt{cv_broadcast}.

CV_TIMEDOUT  The condition was not necessarily signaled because the timeout expired.

CONSTRAINTS

Must be called in thread context.

The order of spinlocks passed to \texttt{cv_timedwait} must be less than \texttt{CV_SPINLOCK_ORDER}.

EXAMPLE

#include <sys/condvar.h>

cv_t *queue_cvp;
/*
 * Mutex that protects the queue of resources.
 */
mutex_t *queue_mtxp;
int error;

int get_resource(resource_t *resource, timestruc_t *timep);
{
  /*
   * Get next resource off the queue.
   */
  mutex_lock (queue_mtxp);
  while (queue_is_empty ()) {
    error = cv_timedwait(queue_cvp, queue_mtxp, CV_MUTEX,
                         CV_DFLT_FLG, timep);
    if (error == CV_SIGNALED) {
      /*
       * Must unlock before return.
       */
      mutex_unlock(queue_mtxp);
      return(error);
    }
  }
  *resource = dequeue_head_of_list ();
  mutex_unlock (queue_mtxp);
  return (OK);
}

SEE ALSO

cv_alloc(9F), cv_attr_init(9F), cv_attr_setdata(9F), cv_broadcast(9F), cv_dealloc(9F), cv_signal(9F),
cv_timedwait_sig(9F), cv_wait(9F), cv_wait_sig(9F)
NAME

cv_timedwait_sig - Wait for the condition associated with a condition variable or a timeout

SYNOPSIS

#include <sys/condvar.h>

int cv_timedwait_sig(
    cv_t *cvp,
    void *lock_p,
    cv_lock_type_t lt,
    cvw_flags_t flags,
    timestruc_t *timep
);

PARAMETERS

cvp Pointer to the condition variable upon which to wait.
lockp Pointer to the lock protecting the condition variable.
lt Specifies the type of lock. This can be a spinlock, mutex, reader-writer lock, or reader-writer spinlock.
flags Specifies the options or modifiers to the function. Must be the logical OR of one or more of the flags defined by the cvw_flags_t enumerated data type. See the DESCRIPTION section for more information.
timep The length of time, in seconds and nanoseconds, to wait.

DESCRIPTION

The cv_timedwait_sig routine waits for the occurrence of a condition associated with a condition variable, an HP-UX signal, or a timeout. It atomically blocks the calling thread and unlocks the lock; this guarantees that the thread will not miss a condition change. The cv_timedwait_sig routine returns to the thread with the lock locked, unless CV_NO_RELOCK is specified.

The thread must be able to handle spurious wakeups. There is no guarantee that the condition is true when the thread resumes execution. If the wait is interrupted by an HP-UX signal, the thread must tell the caller that an interrupt occurred. This information must be passed up the call stack until the thread returns to user space in order for the application signal handler to be invoked.

If a timer value is specified that requires finer granularity than the system supports, the timer is rounded up to a supported value.

The cv_lock_type_t enumerated data type defines the following lock types:

CV_MUTEX The lock is an adaptive mutex.
CV_REC_MUTEX The lock is a recursive mutex lock.
CV_RWLOCK_RD The lock is a read lock.
CV_RWLOCK_WR The lock is a write lock.
CV_RWSPIN_RD The lock is a read spinlock. You can only lock the lock one time.
CV_RWSPIN_WR The lock is a write spinlock. You can only lock the lock one time.
CV_SPIN The lock is a spinlock.

The cvw_flags_t enumerated data type defines the following flags:

CV_DFLT_FLG The default value for the flag. Use this value if no other options are specified.
CV_NO_RELOCK  Do not relock the lock before returning to the user. After the lock is unlocked, the cv_wait routines will not access the condition variable.

Threads are inserted in the sleep queue based on their scheduling priority or PZERO-1, whichever is greater, unless the condition variable was created with the CV_ATTR_FIFO option.

RETURN VALUES

CV_INTR  The condition was not necessarily met and the function returned because an HP-UX signal is pending

CV_SIGNALED  The condition was met and the function returned because of a call to cv_signal or cv_broadcast.

CV_TIMEDOUT  The condition was not necessarily met because the timeout expired.

CONSTRAINTS

Must be called in thread context.

The deadlock order of spinlocks passed to cv_timedwait_sig must be less than CV_SPINLOCK_ORDER.

EXAMPLE

#include <sys/condvar.h>

cv_t *queue_cvp;
/*
 * Mutex that protects the queue of resources.
 */
mutex_t *queue_mtxp;

int get_resource(resource_t *resource, timestruc_t *timep);
{
    /*
    * Get next resource off the queue.
    */
    mutex_lock (queue_mtxp);
    while (queue_is_empty ()) {
        error = cv_timedwait_sig(queue_cvp, queue_mtxp, CV_MUTEX,
                                CV_DFLT_FLG, timep);
        if (error == CV_SIGNALED) {
            /*
            * Must unlock before return.
            */
            mutex_unlock(queue_mtxp);
            return(error);
        }
    }
    *resource = dequeue_head_of_list ();
    mutex_unlock (queue_mtxp);
    return (OK);
}

SEE ALSO

cv_alloc(9F), cv_attr_init(9F), cv_attr_setdata(9F), cv_broadcast(9F), cv_dealloc(9F), cv_signal(9F),
cv_timedwait(9F), cv_wait(9F), cv_wait_sig(9F)
cv_wait(9F)

NAME

cv_wait - Wait for the condition associated with a condition variable

SYNOPSIS

#include <sys/condvar.h>

void cv_wait(
    cv_t *cvp,
    void *lock_p,
    cv_lock_type_t lt,
    cvw_flags_t flags
);

PARAMETERS

cvp Pointer to the condition variable upon which to wait.
lockp Pointer to the lock protecting the condition variable.
lt Specifies the type of lock. This can be a spinlock, mutex, reader-writer lock, or reader-writer spinlock.
flags Specifies the options or modifiers to the function. Must be the logical OR of one or more of the flags defined by the cvw_flags_t enumerated data type. See the DESCRIPTION section for more information.

DESCRIPTION

The cv_wait routine waits for the occurrence of the condition associated with the condition variable. It atomically blocks the calling thread and unlocks the lock; this guarantees that the thread will not miss a condition change. The thread remains blocked until another thread calls either cv_signal or cv_broadcast. An HP-UX signal will not unblock the thread. The cv_wait routine returns to the thread with the lock locked, unless CV_NO_RELOCK is specified. If the lock is a reader-writer lock or reader-writer spinlock, the lock is locked in the same mode (read or write) as it was upon entry.

The thread must be able to handle spurious wakeups. There is no guarantee that the condition is true when the thread resumes execution.

The cv_lock_type_t enumerated data type defines the following lock types:

CV_MUTEX The lock is an adaptive mutex.
CV_REC_MUTEX The lock is a recursive mutex lock.
CV_RWLOCK_RD The lock is a read lock.
CV_RWLOCK_WR The lock is a write lock.
CV_RWSPIN_RD The lock is a read spinlock. You can only lock the lock one time.
CV_RWSPIN_WR The lock is a write spinlock. You can only lock the lock one time.
CV_SPIN The lock is a spinlock.

The cvw_flags_t enumerated data type defines the following flags:

CV_DFLT_FLG The default value for the flag. Use this value if no other options are specified.
CV_NO_RELOCK Do not relock the lock before returning to the user. After the lock is unlocked, the cv_wait routines will not access the condition variable.

Threads are inserted in the sleep queue based on their scheduling priority or PZERO-1, whichever is greater, unless the condition variable was created with the CV_ATTR_FIFO option.
RETURN VALUES

None

CONSTRAINTS

Must be called in thread context.
The order of spinlocks passed to cv_wait must be less than CV_SPINLOCK_ORDER.

EXAMPLE

#include <sys/condvar.h>

cv_t *queue_cvp;
/*
 * Mutex that protects the queue of resources.
 */
mutex_t *queue_mtxp;
/*
 * Get next resource off the queue.
 */
mutex_lock (queue_mtxp);
while (queue_is_empty ()) {
    cv_wait(queue_cvp, queue_mtxp, CV_MUTEX, CV_DFLT_FLG);
}
resource = dequeue_head_of_list ();
mutex_unlock (queue_mtxp);

SEE ALSO

cv_alloc(9F), cv_attr_init(9F), cv_attr_setdata(9F), cv_broadcast(9F), cv_dealloc(9F), cv_signal(9F),
cv_timedwait(9F), cv_timedwait_sig(9F), cv_wait_sig(9F)
NAME

cv_wait_sig - Wait for the condition associated with a condition variable

SYNOPSIS

#include <sys/condvar.h>

int cv_wait_sig(
    cv_t *cvp,
    void *lock_p,
    cv_lock_type_t lt,
    cvw_flags_t flags
);

PARAMETERS

cvp Pointer to the condition variable upon which to wait.
lockp Pointer to the lock protecting the condition variable.
l Specifies the type of lock. This can be a spinlock, mutex, reader-writer lock, or reader-writer spinlock.
flags Specifies the options or modifiers to the function. Must be the logical OR of one or more of the flags defined by the cvw_flags_t enumerated data type. See the DESCRIPTION section for more information.

DESCRIPTION

The cv_wait_sig routine waits for the occurrence of an HP-UX signal to interrupt the thread’s wait. It atomically blocks the calling thread and unlocks the lock; this guarantees that the thread will not miss the signaling of a condition change. The cv_wait_sig routine returns to the thread with the lock locked, unless CV_NO_RELOCK is specified.

The thread must be able to handle spurious wakeups. There is no guarantee that the condition is true when the thread resumes execution. If the wait is interrupted by an HP-UX signal, the thread must tell the caller that an interrupt occurred. This information must be passed back up the call stack until the thread returns to user space in order for the application signal handler to be invoked.

The cv_lock_type_t enumerated data type defines the following lock types:

CV_MUTEX The lock is an adaptive mutex.
CV_REC_MUTEX The lock is a recursive mutex lock.
CV_RWLOCK_RD The lock is a read lock.
CV_RWLOCK_WR The lock is a write lock.
CV_RWSPIN_RD The lock is a read spinlock. You can only lock the lock one time.
CV_RWSPIN_WR The lock is a write spinlock. You can only lock the lock one time.
CV_SPIN The lock is a spinlock.

The cvw_flags_t enumerated data type defines the following flags:

CV_DFLT_FLG The default value for the flag. Use this value if no other options are specified.
CV_NO_RELOCK Do not relock the lock before returning to the user. After the lock is unlocked, the cv_wait routines will not access the condition variable.

Threads are inserted in the sleep queue based on their scheduling priority or PZERO-1, whichever is greater, unless the condition variable was created with the CV_ATTR_FIFO option.
RETURN VALUES

CV_INTR The condition was met and the function returned because of a call to cv_signal or cv_broadcast.

CV_SIGNALED The function returned because an HP-UX signal is pending.

CONSTRAINTS

Must be called in thread context.

The deadlock order of spinlocks passed to cv_wait_sig must be less than CV_SPINLOCK_ORDER.

EXAMPLE

```c
#include <sys/condvar.h>

cv_t *queue_cvp;
/*
 * Mutex that protects the queue of resources.
 */
mutex_t *queue_mtxp;
int error;

int get_resource(resource_t *resource);
{
    /*
    * Get next resource off the queue.
    */
    mutex_lock (queue_mtxp);
    while (queue_is_empty ()) {
        error = cv_wait_sig(queue_cvp, queue_mtxp, CV_MUTEX, CV_DFLT_FLG);
        if (error == CV_INTR) {
            /*
            * Must unlock before return.
            */
            mutex_unlock(queue_mtxp);
        }
    }
    *resource = dequeue_head_of_list ();
    mutex_unlock (queue_mtxp);
    return (OK);
}
```

SEE ALSO

cv_alloc(9F), cv_attr_init(9F), cv_attr_setdata(9F), cv_broadcast(9F), cv_dealloc(9F), cv_signal(9F),
 cv_timedwait(9F), cv_timedwait_sig(9F), cv_wait(9F)
NAME

delay, delay_sig - Suspend the calling thread for a specified number of clock ticks.

SYNOPSIS

#include <sys/kern_svc.h>
#include <sys/param.h>

void delay(
    clock_t ticks
);

int delay_sig(
    clock_t ticks
);

PARAMETERS

ticks Specifies time value in number of clock ticks. If ticks is zero, the function returns immediately.

DESCRIPTION

The delay kernel function suspends the calling thread for ticks number of clock ticks. The thread sleeps in a non-interruptible mode during this time, and cannot be interrupted by signals until the specified time has expired.

The delay_sig kernel function also suspends the calling thread for ticks number of clock ticks, but it sleeps interruptibly. The delay_sig function indicates through a return value if it was interrupted by a signal.

The Hz value defined in <sys/param.h> file defines the number of clock ticks per second.

Although ticks is defined to be type clock_t, the constant MAXINT defined in <sys/param.h> specifies the actual limit on the number of ticks that a thread can sleep. If ticks is greater than this maximum, it is silently rounded down to it.

The delay and delay_sig functions do not busy wait. Use the busywait function if busy waiting is needed.

Use the timeout function to asynchronously execute some function after a delay of specified number of clock ticks.

RETURN VALUES

delay Has no return value.

delay_sig Returns the following values:

  0 The sleep completed normally and was not interrupted.

  EINTR The sleep was interrupted by a signal.

CONSTRAINTS

Do not call while holding any spinlocks.

Do not call from interrupt context.

EXAMPLES

/*
  * Block this thread for three seconds, and then resume.
  */

delay(3*HZ);
SEE ALSO

busywait(9F), timeout(9F), untimeout(9F).
disksort_dequeue(9F)

NAME

disksort_dequeue – Dequeue a buffer from the disksort queue.

SYNOPSIS

#include <sys/disksort.h>

struct buf* disksort_dequeue(
 struct buf *dp
);

PARAMETERS

dp Pointer to the head of the disksort queue.

DESCRIPTION

The disksort_dequeue function dequeues a buffer from the disksort queue pointed to by dp. The enqueued buffers are sorted in an order that attempts to minimize seek and rotational latencies experienced by the disk device. The order in which buffers are dequeued may differ from the order in which the driver enqueued the buffers.

The disk driver is responsible for synchronizing access to the disksort queue. The disk driver must use a lock, such as a spinlock, when calling disksort_enqueue and disksort_dequeue.

RETURN VALUES

<>NULL Pointer to the buffer dequeued.

NULL No buffer dequeued.

CONSTRAINTS

May be called in user context or interrupt context.

May be called while holding a spinlock.

EXAMPLE

lock_t *mylock = abc; /* Pointer to driver spinlock */
struct buf *dp = xyz; /* Pointer to head of disksort queue */
struct buf *bp; /* Pointer to buffer dequeued */

...

spinlock(mylock);
bp = disksort_dequeue(dp);
spinunlock(mylock);
if (bp != NULL)
{
  /* Issue the buffer dequeued to the disk device. */
  
  /* */
}

SEE ALSO

disksort_enqueue(9F), disksort_init_queue(9F)
disksort_enqueue(9F)

NAME

disksort_enqueue - Enqueue a buffer onto the disksort queue.

SYNOPSIS

#include <sys/disksort.h>

void disksort_enqueue(
    struct buf *dp,
    struct buf *bp
);

PARAMETERS

dp  Pointer to the head of the disksort queue.
bp  Pointer to the buffer to be enqueued.

DESCRIPTION

The disksort_enqueue function enqueues the buffer onto the disksort queue pointed to by dp. The enqueued buffers are sorted in an order that attempts to minimize seek and rotational latencies experienced by the disk device. The order in which buffers are dequeued might differ from the order in which the driver enqueued the buffers.

The disk driver is responsible for synchronizing access to the disksort queue. The disk driver must hold a lock, such as a spinlock, when calling disksort_enqueue and disksort_dequeue.

CONSTRAINTS

May be called in user context or interrupt context.
May be called while holding a spinlock.

EXAMPLE

mydriver_strategy(struct buf *bp)
{
    lock_t *mylock = abc;    /* Pointer to driver spinlock */
    struct buf *dp = xyz;    /* Pointer to head of disksort queue */
    ...
    spinlock(mylock);
    disksort_enqueue(dp, bp);
    spinunlock(mylock);
}

SEE ALSO

disksort_dequeue(9F), disksort_init_queue(9F)
**NAME**

disksort_init_queue - Initialize the disksort queue.

**SYNOPSIS**

```c
#include <sys/disksort.h>
void disksort_init_queue(
    struct buf *dp
);
```

**PARAMETERS**

`dp`  Pointer to the head of the disksort queue.

**DESCRIPTION**

The `disksort_init_queue` function initializes the `buf` structure defining the `disksort` queue. The queue contains a sorted list of buffers that have been enqueued by a disk driver instance. The disk driver must allocate storage for the `buf` structure, pointed to by `dp`, that is used as the head of the `disksort` queue.

**CONSTRAINTS**

May be called in user context or interrupt context.

May be called while holding a spinlock.

**EXAMPLE**

```c
static int
mydriver_open(dev_t dev, int oflags)
{
    struct buf *dp = xyz;  /* Pointer to head of disksort queue */
    ...
    if (is_first_open) {
        disksort_init_queue(dp);
    }
    ...
}
```

**SEE ALSO**

`disksort_dequeue(9F)`, `disksort_enqueue(9F)`
NAME
FREE - Kernel macro to deallocate kernel memory.

SYNOPSIS
#include <sys/malloc.h>
#define FREE (addr, type)

PARAMETERS
addr Address of the kernel memory previously allocated by MALLOC.
type The memory allocation pool type.

DESCRIPTION
The MALLOC and FREE kernel macros are deprecated interfaces and may be obsoleted in a future release of HP-UX. Use the kernel functions kmalloc and kfree in place of MALLOC and FREE. The FREE kernel macro deallocates (frees) kernel memory previously allocated by MALLOC. The addr parameter must be the address returned by MALLOC and type must be the same type passed to MALLOC.

CONSTRAINTS
None

EXAMPLES
struct buf * bp;

/*
 * Allocate a buf structure for internal use.
 */
MALLOC(bp, struct buf *, sizeof(struct buf),
       M_IOSYS, M_NOWAIT);
...

/*
 * When done with the buf structure, release it.
 */
FREE(bp, M_IOSYS);

SEE ALSO
kmalloc(9F), kfree(9F), MALLOC(9F)
**NAME**

getc - Manipulate characters on a clist.

**SYNOPSIS**

#include <sys/clist.h>

int getc(
    struct clist *list
);

**PARAMETERS**

*list*  Pointer to a clist.

**DESCRIPTION**

The `getc` kernel function returns the next character in the clist *list*.

**RETURN VALUES**

* The next character in the clist.
-1 If no characters are in the clist.

**CONSTRAINTS**

None

**EXAMPLES**

```c
void catq(struct clist * from, struct clist * to)
    /* concatenate characters from one clist to another */
{
    int c;

    while ((c = getc(from)) >= 0)
        (void) putc(c, to);
}
```

**SEE ALSO**

`putc(9F), getcb(9F), getcf(9F), putcb(9F), putcf(9F)`, `putc(9F), getcb(9F), getcf(9F), putcb(9F), putcf(9F)`
**NAME**
getcb - Manipulate cblocks on a clist.

**SYNOPSIS**
#include <sys/clist.h>
struct cblock *getcb(
    struct clist *list
);

**PARAMETERS**
list Pointer to a clist.

**DESCRIPTION**
The getcb removes the last cblock from the clist pointed to by list.

**RETURN VALUES**
<>NULL Successful completion. The value is a pointer to the cblock that was removed.
NULL Error.

**CONSTRAINTS**
None

**EXAMPLE**
int
mydev_open(dev_t dev, int flag)
{
    struct cblock *cp;
    struct mydevice dv;

    /* Flush out the device buffer */
    while ((cp = getcb(&dv->queue)) != NULL)
        putcf(cp);

    ...
}

**SEE ALSO**
putcb(9F), getc(9F), getcf(9F), putc(9F), putcf(9F)
**NAME**

getcf - Get a cblock on the cblock free list.

**SYNOPSIS**

```c
#include <sys/clist.h>
struct cblock *getcf(void);
```

**PARAMETERS**

None

**DESCRIPTION**

The `getcf` function gets the next free `cblock` on the `cfreelist`.

**RETURN VALUES**

<>NULL Successful completion. The value is a pointer to the `cblock`.
NULL Error.

**CONSTRAINTS**

None

**SEE ALSO**

`getc(9F), getcb(9F)`
geteblk(9F)

NAME
geteblk - Allocate a buffer from the buffer cache.

SYNOPSIS
#include <sys/buf.h>

struct buf *geteblk(
    int size
);

PARAMETERS
size  The size in bytes of the requested block.

DESCRIPTION
The geteblk kernel function allocates a buffer large enough to contain size bytes of data from
the buffer cache and returns a pointer to the buffer header. The size parameter must be <=
MAXBSIZE defined in <sys/param.h>.

Fields in the buf structure returned are initialized as follows:
- b_flags is set with B_BUSY, B_BCACHE, and B_INVAL flags.
- b_un.b_addr contains the virtual address of the allocated buffer.
- b_spaddr contains the space address (space ID) of the allocated buffer.

Drivers do not normally allocate buffers. However, to implement special features, such as ioctl
commands, the driver may need its own buffer space. Buffers allocated by calling geteblk must
be used for short durations and released by calling brelse when no longer needed.

RETURN VALUES
The geteblk returns a pointer to the allocated buffer header.

CONSTRAINTS
Do not call in an interrupt context.
Do not call while holding a spinlock.

EXAMPLE
struct buf * bp;

/*
 * Allocate a buffer that can store a page of data.
 */
bp = geteblk(PAGESIZE);
VASSERT(bp);

SEE ALSO
brelse(9F), buf(9F)
geterror(9F)

NAME

gterror - Return the error number from the buffer header.

SYNOPSIS

#include <sys/buf.h>

int geterror(
  struct buf *bp
);

PARAMETERS

bp  Pointer to a buf structure.

DESCRIPTION

The geterror returns the error number from the b_error field of the buffer header structure pointed to by bp. If B_ERROR is not set in bp->b_flags, the I/O is assumed to have completed successfully.

RETURN VALUES

0   I/O completed successfully.
<0  Error number.

CONSTRAINTS

None

EXAMPLE

int error;
struct buf *bp;
...

/*
 * After starting the I/O request, wait for its completion.
 */
error = biowait(bp);

/*
 * biowait returns 0 if the IO completes successfully.
 * A non-zero value is returned if an error has been
 * encountered, however, the error value returned is not
 * always for the IO completion. To get the IO
 * completion error that is returned with the buf, we
 * need to call geterror().
 */
if (error) {
   error = geterror(bp);
}

SEE ALSO

biowait(9F), buf(9F)
get_system_time(9F)

NAME

get_system_time - Get the current date and time with accuracy to 1/100 second

SYNOPSIS

#include <sys/kernel.h>
#include <sys/_timeval.h>

struct timeval get_system_time(void);

DESCRIPTION

The get_system_time function returns the current time, expressed as seconds and microseconds since 00:00 Coordinated Universal Time (UTC), January 1, 1970. The accuracy of 1/100 second is based on the kernel global value of hz, which is typically 100.

RETURN VALUES

On successful return, timeval contains the current number of seconds and microseconds that have elapsed since 00:00 UTC. January 1, 1970. If this function is called before the machdep clock subsystem initialization, timeval is 0.

CONSTRAINTS

Can be called in interrupt context.
Can be called while holding a spinlock.

EXAMPLES

#include <sys/kernel.h>       /* for get_system_time prototype */
#include <sys/_timeval.h>  /* for struct timeval definition */

/*
 * Obtain the current time
 */

struct timeval current_time = get_system_time();

SEE ALSO

ms_gettimeofday(9F)
NAME

gsignal - Send the specified signal to all processes in a process group.

SYNOPSIS

#include <sys/proc_iface.h>
void gsignal(
    pid_t pgrp,
    int sig
);

PARAMETERS

pgrp  Process group identifier.
sig   Signal number.

DESCRIPTION

The gsignal kernel function sends the signal sig to all processes that have pgrp as their process group identifier.

The pgrp parameter is returned by the p_grp kernel function.

Signal numbers are defined in <sys/signal.h>.

RETURN VALUES

None

CONSTRAINTS

This interface cannot be called from Interrupt Control Stack (ICS).

EXAMPLE

#include <sys/proc.h>
#include <sys/signal.h>
#include <sys/proc_iface.h>

/*
 * Save the pointer to the proc structure
 */
proc_t *my_procp = procp_self();
/*
 * Signal all processes in the current process group.
 */
gsignal(p_pgrp(my_procp), SIGIO);

SEE ALSO

p_pgrp(9F), psignal(9F)
NAME

iomap_enable_wc - Enables write coalescing for a range of device I/O memory.

SYNOPSIS

#include <sys/iomap.h>

int iomap_enable_wc(
    space_t space,
    void *vaddr,
    pgcnt_t count
);

PARAMETERS

space The vaddr space identifier.
vaddr The virtual address within the I/O range.
count Count of base pages within the I/O range.

DESCRIPTION

The iomap_enable_wc function enables the WC (write coalescing) memory attribute, if supported by the platform hardware, for memory mapped I/O pages starting at the virtual address vaddr for count pages. The vaddr value must be aligned on a base page and within the virtual address range returned by kernel_iomap, user_iomap, or user_iomap_private.

You can call iomap_enable_wc after a range of I/O pages has been memory mapped (by calls to kernel_iomap, user_iomap, or user_iomap_private) and before the first access to the memory mapped I/O range. The vaddr and count parameters might be a subset of pages within the range of pages that were previously memory mapped.

After each call to an I/O mapping function for a range of I/O pages for which you want to enable write coalescing, HP recommends that you call iomap_enable_wc. This ensures that aliased virtual addresses to the same I/O pages are assigned the same memory attributes. If WC is assigned to only one of the aliased virtual addresses, platform behavior is undefined.

NOTE: A call to user_iomap for a 32-bit user process and a call to user_iomap for a 64-bit process to the same I/O pages might return different virtual addresses that are aliased to the same I/O pages.

After WC is enabled for an I/O page, you cannot disable it while the mapping exists. If a memory mapped I/O page on which WC is enabled is unmapped (by a call to user_iounmap), the caller must ensure the WC buffer is flushed before unmapping the I/O page.

RETURN VALUES

0 WC successfully enabled.
WC_NOT_SUPPORTED Platform does not support the WC attribute.
WC_NOT_IOSPACE One or more pages are not in I/O space.

CONSTRAINTS

Do not call in interrupt context.
Do not call while holding a spinlock.
EXAMPLES

void  *physaddr;
void  *useraddr;
size_t pagesize;

/*
 * Get the physical address as seen by the processor.
 * Verify that the physical address is pagesize aligned.
 */
physaddr = xyz;
pagesize = iomap_pagesize();
VASSERT(((uintptr_t)physaddr % pagesize) == 0);

/*
 * Memory map the user virtual address onto a page of
 * I/O memory.
 */
useraddr = user_iomap(NULL, physaddr, 1L);

if (useraddr) {
    wc_status_t status;
    /*
     * Enable write coalescing (WC) for the mapped I/O page
     */
    status = iomap_enable_wc(ldusid(useraddr), useraddr, 1L);
} else {
    /*
     * Failed to map user virtual address.
     * Add error handling code here.
     */
}

SEE ALSO

kernel_iomap(9F), kernel_iounmap(9F), user_iomap(9F), user_iomap_private(9F), user_iounmap(9F).
NAME

iomap_pagesize - Return pagesize in bytes for memory mapped I/O.

SYNOPSIS

#include <sys/iomap.h>

size_t iomap_pagesize(void);

PARAMETERS

None

DESCRIPTION

The iomap_pagesize function returns the pagesize in bytes for memory mapped I/O.

RETURN VALUES

Page size.

CONSTRAINTS

May be called in user or interrupt context.
May be called while holding spinlocks.

EXAMPLE

size_t pagesize;

/*
 * Verify the I/O physical address is aligned on a
 * page boundary in memory mapped I/O address space.
 */
pagesize = iomap_pagesize();
VASSERT(((uintptr_t)iophysaddr % pagesize) == 0);

SEE ALSO

kernel_iomap(9F), kernel_iounmap(9F), user_iomap(9F), user_iounmap(9F).
NAME

iovec - Data buffer descriptor for character I/O and WSIO CDIO mapping services.

SYNOPSIS

#include <sys/uio.h>

DESCRIPTION

The iovec kernel structure points to the data buffer for character I/O. It can be declared and used to map a kernel data buffer for an I/O bus master and it is a member of the uio structure used by uiomove.

STRUCTURE MEMBERS

The iovec structure is defined in <sys/uio.h>. It has the following fields. Their data types are shown in the table below. Drivers must not modify any fields of this structure when using uiomove. When this structure is used for mapping with a WSIO CDIO mapping service, the driver initially sets up iov_base and iov_len.

Table 1-2 iovec Structure Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>iov_base</td>
<td>caddr_t</td>
</tr>
<tr>
<td>iov_len</td>
<td>size_t</td>
</tr>
</tbody>
</table>

- **iov_base**  The address of the user’s buffer for uiomove or kernel buffer for WSIO CDIO mapping services.
- **iov_len**  The number of bytes to be transferred.

SEE ALSO

uio(9F), wsio_fastmap(9F), wsio_map(9F), wsio_remap(9F)
**NAME**

kdaemon_proc_create - Create a kernel daemon process.

**SYNOPSIS**

```c
#include <sys/kdaemon_proc.h>

int kdaemon_proc_create(
    pid_t *cpid,
    void (*start_func)(void *),
    void *arg,
    char *proc_name,
    uint64_t flags
);
```

**PARAMETERS**

- **cpid**  
  Pointer to the location where the created kernel daemon process ID is to be returned.

- **start_func**  
  Function to be executed by the created kernel daemon process.

- **arg**  
  Parameter to be passed to the created kernel process' start_func function.

- **proc_name**  
  Kernel daemon process name string. There is an implementation-defined maximum string length. The string must be null terminated.

- **flags**  
  Flags to indicate characteristics of the created kernel daemon process. If the value is zero, the default characteristics are used.

**DESCRIPTION**

Use the kdaemon_proc_create function to create a new kernel daemon process. The created process begins its execution at start_func. A return from start_func results in an implicit exit of the kernel daemon process as if kdaemon_proc_exit was called. The new process is created with exactly one kernel thread.

The new kernel daemon process does not follow the fork(2) inheritance protocol from user space. The following attributes are not inherited, but initialized to defaults:

- Real, effective, and saved user ID's are set to root.
- Real, effective, and saved group ID's are set to root.
- Process group ID is set to root.
- No file descriptors.
- Does not process signals.
- Profiling is disabled.
- No accounting.
- No attached shared memory segments.
- No current working directory.
- Default file mode creation mask.
- No file size limit.
- Scheduling policy is initially set to system priority range to prevent the created kernel daemon process from monopolizing CPU time. If needed, use sched_set_pripolicy to set different scheduling policy or priority.

Multithreaded kernel daemon processes cannot create a new kernel daemon process because they might never enter clean points in the kernel or might never sleep interruptibly to allow suspension of the kernel threads.
RETURN VALUES

0       A new kernel daemon process was successfully created.
EINVAL  start_func is specified as NULL, proc_name is specified as NULL, pid is specified as NULL, or the specified flags are invalid.
EPERM   A multithreaded kernel daemon process is trying to create a new kernel daemon process.
EAGAIN  The system-imposed limit on the total number of processes under execution has been exceeded.
ENOMEM  There is insufficient physical memory available in the system to create a new kernel daemon process.

CONSTRAINTS

Do not call while holding a spinlock.
Do not call in an interrupt context.

EXAMPLES

void
ex ample_func(void *arg)
{
    while(need_to_do_some_work) {
        do_work();
    }
    /*
    * Must be in clean point (not holding any kernel
    * resource).
    */
    kthread_proc_exit(0L);
}

pid_t cpid;
    retval = kdaemon_proc_create(&cpid, example_func, (void *)NULL,
                 "Example Proc", (uing64_t)0);

if (retval > 0) {
    printf("kdaemon_proc_create() failed with error %d\n",
        retval);
} else {
    printf("kdaemon_proc_create() created a new proc with pid %d\n",
          pid);
}

SEE ALSO

kdaemon_proc_exit(9F), kdaemon_thread(9F)
NAME
kdaemon_proc_exit - Terminate a kernel daemon process.

SYNOPSIS
#include <sys/kdaemon_proc.h>
void kdaemon_proc_exit(
    long status
);

PARAMETERS
status The calling process' exit status.

DESCRIPTION
The kdaemon_proc_exit function terminates the calling kernel daemon process. If the kernel
daemon process is multithreaded, all threads in the process are terminated.
The routine does not return to its caller. Therefore, the caller must ensure that it is not holding
any system resources. Calling kthread_proc_exit while holding kernel resource is undefined
and can lead to a system hang.

RETURN VALUES
None

CONSTRAINTS
Do not call while holding locks of any kind, including spinlocks and blocking locks.
Do not call in an interrupt context.
Must be called from a clean point in the process' execution; do not hold any kernel resources.

EXAMPLES
kdaemon_func(void *arg)
{
    do_work();
    kdaemon_proc_exit(0L);
    /*
     * Never reaches this point.
     */
}

SEE ALSO
kdaemon_proc_create(9F), kdaemon_thread(9F)
NAME

kdaemon_thread - Kernel daemon thread routines.

SYNOPSIS

#include <sys/kdaemon_thread.h>

int kdaemon_thread_create(
    void (*func) (void *),
    void *data,
    struct proc *p,
    tid_t *newtid,
    uint64_t flags
);

void kdaemon_thread_exit(void);

int kdaemon_thread_wait(
    tid_t wait_for,
    tid_t *waited_for
);

DESCRIPTION

Kernel daemon threads, typically created at system initialization or DLKM initialization, have no user context and run entirely within the kernel. In addition, they cannot respond to signals. The kdaemon_thread_* API functions allow callers to create, manage, and destroy kernel daemon threads.

The kdaemon_thread_* functions are intended to operate only upon kernel daemon threads. Error returns or unspecified behavior might result if the kdaemon_thread_* functions are used to act upon threads other than kernel daemon threads.

Summary of Available Calls

kdaemon_thread_create Creates a new kernel daemon thread within the specified process. The new kernel daemon thread begins execution at its start function.

kdaemon_thread_exit Causes the calling kernel daemon thread to exit.

kdaemon_thread_wait Allows a thread to wait for another kernel daemon thread.

See the individual kdaemon_thread_*(9F) manpages for a detailed description of each function.

EXAMPLES

/*
 * This code snippet has a main thread create a worker thread, which
 * does its work and then exits. The main thread then waits upon the
 * worker thread. The process p is a kernel daemon process.
 */
#include <sys/kdaemon_thread.h>

/*
 * Set up a worker thread to do work.
 */
ret = kdaemon_thread_create(work_thread,(void *)work,p,&newtid,
               (uint64_t)0);

if (ret != 0) {
    return ret;
ret = kdaemon_thread_wait(newtid,NULL);
if (ret != 0) {
    return ret;
}

static void
work_thread(void *work)
{
    /*
     *  perform work
     */
    kdaemon_thread_exit();
}

SEE ALSO
kdaemon_thread_create(9F), kdaemon_thread_exit(9F), kdaemon_thread_wait(9F).
NAME

kdaemon_thread_create - Create a kernel daemon thread.

SYNOPSIS

#include <sys/kdaemon_thread.h>

int kdaemon_thread_create(
    void (*start_func) (void *),
    void *arg,
    struct proc *p,
    tid_t *newtid,
    uint64_t flags
);

PARAMETERS

start_func Function in which the newly created kernel daemon thread will run. A return from this function will result in the termination of the kernel daemon thread created using this routine.

arg The data passed as an argument to start_func upon startup of the newly created kernel daemon thread.

p Process in which the newly created kernel daemon thread will be contained. This process can be a daemon process or a user process. However, if p is a user process, the calling process must also be p.

newtid Pointer to address in which the thread ID of the newly created kernel daemon thread will be placed.

flags The following values may be OR'd together to specify the creation options for the created thread:

KDAEMON_THREAD_CREATE_DETACHED Create a detached thread. This flag is automatically set if the process p is not a kernel daemon process.

KDAEMON_THREAD_CREATE_NO_WAIT Create the thread only if it can be done without sleeping.

KDAEMON_THREAD_CREATE_NO_INTR_WAIT Create the thread only if it can be done without sleeping interruptibly.

DESCRIPTION

Create a kernel daemon thread in a process. New kernel daemon threads can be created in both kernel daemon processes and user processes. Kernel daemon threads, typically created at system initialization or dlkm initialization, have no user context and run entirely within the kernel. Kernel daemon threads cannot respond to signals.

NOTE: Kernel daemon threads created with this routine in user processes are created detached. If a kernel daemon thread is created in a different kernel daemon process without the KDAEMON_THREAD_CREATE_DETACHED flag set, that kernel daemon process must make arrangements to cleanup that kernel daemon thread's resources if it exits.

The creation of kernel daemon threads in a user process different from the caller's process is not supported.
RETURN VALUES

0 Success

EBUSY KDAEMON_THREAD_CREATE_NO_WAIT specified and the calling thread would have had to sleep waiting for a resource needed to complete this operation.

EBUSY KDAEMON_THREAD_CREATE_NO_INTR_WAIT specified and the calling thread would have had to sleep interruptibly waiting for a resource needed to complete this operation.

EINTR KDAEMON_THREAD_CREATE_NO_WAIT was not specified and a sleep waiting for a resource needed to complete this routine was interrupted.

EAGAIN The maximum number of threads in the containing process would be exceeded by a successful call to this routine.

ENOMEM No memory left to allocate space for a new kernel daemon thread.

EINVAL Invalid input parameters: Invalid flags, p is not a proc, newtid is NULL, or start_func is NULL.

EPERM An attempt was made to create a kernel daemon thread in user process p that is different from the calling process.

CONSTRAINTS

Do not call while holding a spinlock.

Do not call in an interrupt context.

EXAMPLES

/*
 * Create a thread that performs work and then exits implicitly.
 */
#include <sys/kdaemon_thread.h>
...
ret = kdaemon_thread_create(async_func,(void *)data,p,&newtid,
KDAEMON_THREAD_CREATE_DETACHED);
if (ret != 0) {
    return ret;
}
...

static void async_func(void *data)
{
    /*
    * Perform work.
    */
}

SEE ALSO

kdaemon_thread(9F), kdaemon_thread_exit(9F), kdaemon_thread_wait(9F).
NAME

kdaemon_thread_exit - Exit the currently running kernel daemon thread.

SYNOPSIS

#include <sys/kdaemon_thread.h>

void kdaemon_thread_exit(void);

PARAMETERS

None

DESCRIPTION

This function causes the calling kernel daemon thread to exit. Only kernel daemon threads are to call this interface; the behavior of this function for non-daemon threads is undefined.

RETURN VALUES

None

CONSTRAINTS

Do not call while holding locks of any kind.
Do not call in interrupt context.

EXAMPLES

/*
 * Within the start function used to start a kernel daemon thread,
 * the function exits when its work is complete.
 */
#include <sys/kdaemon_thread.h>

static void async_thread(void *data)
{
    while (1)
    {
        /*
         * Do new work.
         */
        if (work_done) {
            kdaemon_thread_exit();
        }
    }
}

SEE ALSO

kdaemon_thread(9F), kdaemon_thread_create(9F), kdaemon_thread_wait(9F).
kdaemon_thread_max_thread_proc(9F)

NAME

kdaemon_thread_max_thread_proc - Change the maximum number of threads allowed in a kernel daemon process.

SYNOPSIS

#include <sys/kdaemon_thread.h>

int kdaemon_thread_max_thread_proc(
    struct proc *p,
    int max_threads
);

PARAMETERS

p
    The target kernel daemon process whose maximum thread the caller wants to change.

max_threads
    The maximum number of threads to be allowed in the kernel daemon process p.

DESCRIPTION

The kdaemon_thread_max_thread_proc function allows the maximum number of threads in a kernel daemon process to be changed to the specified number. If the current limit is L1 and we tune it to L2 (L2<L1), we keep the extra (L1-L2) threads but do not allow any additional threads to be created in the process until the current total in use is below L2. A call to this function makes this process immune to any future changes to any system-wide tunables affecting this value.

The max_threads parameter refers to the total number of threads that can be created in the process. In particular, the thread created upon creation of the process counts. For example, if a program needs to create num_helpers helper threads in a process, the maximum number of threads that this process can create must be at least num_helpers+1.

RETURN VALUES

0    Success.

EPERM    The target process is not a kernel daemon process.

EINVAL    The target process p is invalid.

The max_threads parameter is invalid. The value must be between 1 and 1/5 of the total number of threads allowed on the system (if this number is greater than the number of threads currently allowed in this process).

CONSTRAINTS

Do not call while holding blocking locks or spinlocks of order greater than or equal to PROCESS_LOCK_ORDER.

EXAMPLES

/*
 * After learning that we cannot create any more threads, we increase
 * the number and try again.
 */

#include <sys/kdaemon_thread.h>
#include <h/kdaemon_thread.h>

ret = kdaemon_thread_create(func,(void *)NULL,procp,
&newtid, KTHREAD_DAEMON_CREATE_DETACHED);
if (ret == EAGAIN) {
    ret = kdaemon_thread_max_thread_proc(procp, num_i_create+1);
    if (ret == 0) {
        ret = kdaemon_thread_create(func, (void *)NULL, procp,
               &newtid, KTHREAD_DAEMON_CREATE_DETACHED);
    }
}

SEE ALSO
kdaemon_thread_create(9F), kdaemon_thread_kpvt(9F)
kdaemon_thread_needs_to_terminate(9F)

NAME

kdaemon_thread_needs_to_terminate - Determine if there is a request to terminate self.

SYNOPSIS

#include <sys/kdaemon_thread.h>

int kdaemon_thread_needs_to_terminate(void);

PARAMETERS
None

DESCRIPTION

The kdaemon_thread_max_thread_proc function checks to see if there is a request to terminate ourselves. This function is lightweight; it does not block.

This function is intended for kernel daemon threads only; the behavior for non-daemon threads is undefined.

RETURN VALUES

0  There is no request to terminate self.
1  There is a request to terminate self.

CONSTRAINTS

Do not call on ICS.

EXAMPLES

#include <sys/kthread_daemon.h>

... if (kdaemon_thread_needs_to_terminate()) {
    /* Do some cleanup work, then exit. */
    kdaemon_thread_exit();
}...

SEE ALSO

kdaemon_thread_create(9F)
NAME

kdaemon_thread_process_pending_suspension - Processes pending suspension for the calling kernel daemon thread.

SYNOPSIS

#include <sys/kdaemon_thread.h>

void kdaemon_thread_process_pending_suspension(void);

PARAMETERS

None

DESCRIPTION

The kdaemon_thread_process_pending_suspension function processes pending suspensions for the calling kernel daemon thread. It serves as a suspension point for kernel daemon threads. If there is a pending suspension, the calling daemon thread is suspended before eventually returning to the caller. If there is a pending suspension and a pending termination, the calling thread is not suspended; the function only returns.

This function is intended for kernel daemon threads only; the behavior for non-daemon threads is undefined.

RETURN VALUES

None

CONSTRAINTS

Do not call on ICS.
Do not call while holding a spinlock.

EXAMPLES

#include <sys/kthread_daemon.h>
...

/* A typical worker daemon thread pseudocode */

for (;;) {
    kdaemon_thread_process_pending_suspension();
    if (request queue empty) {
        sleep interruptibly waiting for a request;
        if interrupted
            continue;
    }
    dequeue a request;
    if got a request, process it;
}
...

SEE ALSO

kdaemon_thread_create(9F)
kdaemon_thread_wait(9F)

NAME
kdaemon_thread_wait - Wait for a kernel daemon thread in the calling process to exit.

SYNOPSIS
#include <sys/kdaemon_thread.h>

int kdaemon_thread_wait(
    tid_t wait_for,
    tid_t *waited_for
);

PARAMETERS
wait_for Thread ID of the target kernel daemon thread to wait for. A thread ID of 0,
specifiable only within kernel daemon processes, means to wait for any kernel
daoon thread within the calling kernel daemon process to terminate.
waited_for Pointer to an address to store the thread ID of the thread that was waited for.
This is useful if the calling kernel daemon process chose to wait for any kernel
daoon thread (wait_for == 0). This can be NULL if this information is not
desired.

DESCRIPTION
Wait for a kernel daemon thread within the calling process to terminate. You can only wait for
non-detached kernel daemon threads (they were NOT created with the
KDAEMON_THREAD_CREATE_DETACHED flag).

NOTE: Kernel daemon threads created with kdaemon_thread_create in user processes are
automatically be created detached.

If tid is not 0, this call blocks until the kernel daemon thread corresponding to the tid terminates.
If the tid parameter is 0 and the calling process is a kernel daemon process, it waits for the next
thread in the kernel daemon process to terminate. If multiple threads wait for the same tid, at
most one will return successfully, and the others return with an error of ESRCH.

RETURN VALUES
0 Success
ESRCH The kernel daemon thread specified by tid cannot be found in the calling process or
is detached.
EPERM The thread corresponding to the thread ID is not a kernel daemon thread or wait_for
is 0 and the calling process is not a kernel daemon process.
EINTR Wait was interrupted.
EDEADLK Performing this operation would result in system deadlock with this thread waiting
on itself. This can happen, for instance, if a single-threaded kernel daemon process
specifies a tid of 0.

NOTES
You can only wait for kernel daemon threads within your calling process. For example, if a thread
creates a kernel daemon thread in another process, it will not be able to wait for that same kernel
daoon thread.

CONSTRAINTS
Do not call while holding a spinlock.
Do not call in interrupt context.

**EXAMPLES**

```c
/*
 * This code snippet creates a thread and then waits for it.
 */
#include <sys/kdaemon_thread.h>
...
ret = kdaemon_thread_create(func,(void *)data,p,&newtid,
                            (uint64_t)0);
if (ret != 0) {
    return ret;
}

/*
 * Wait for the thread to complete.
 */
ret = kdaemon_thread_wait(newtid,NULL);
if (ret != 0) {
    return ret;
}
...
```

**SEE ALSO**

kdaemon_thread(9F), kdaemon_thread_create(9F), kdaemon_thread_exit(9F).
NAME

kernel_iomap - Map I/O memory to a kernel address.

SYNOPSIS

#include <sys/iomap.h>

void *kernel_iomap(
    void *vaddr,
    void *paddr,
    pgcnt_t count,
    prot_t prot
);

PARAMETERS

vaddr Virtual address requested. must be NULL.
paddr Physical address of the I/O memory location from the view of the processor.
count Count of pages to map.
prot Protection mode for the mapping: PROT_KRW or PROT_URW.

DESCRIPTION

The kernel_iomap function memory maps into kernel address space a range of contiguous pages in I/O address space starting at the physical address paddr for count pages. The value returned is the kernel virtual address that maps onto the base of the I/O memory location specified by paddr and count.

The physical address is the 64-bit physical address from the view of the processor, and the address must be pagesize aligned. You can obtain the pagesize by calling iomappagesize.

The 64-bit physical address (paddr) is not necessarily the same address value that is obtained from the Base Address Register (BAR) of a PCI device. The caller might need to convert the BAR value to the appropriate 64-bit physical address value used by the processor.

NOTE: HP-UX 11i v3 kernels are 64-bit kernels and work with 64-bit physical address values. Although a processor might implement fewer than 64 bits of physical address space, the programming model views the physical address as a 64-bit value.

The prot value specifies the protection mode: PROT_KRW or PROT_URW. If set to PROT_KRW, the kernel is granted access to the I/O memory location, but no user application may be allowed access. If set to PROT_URW, a user may be allowed to share access to the I/O memory location with the kernel. Obtain the user virtual address by calling user_iomap, passing the same paddr and count parameters.

NOTE: Access rights for the returned kernel virtual addresses in HP-UX 11i v3 are always set to PROT_KRW, regardless of the value passed to prot.

To successfully map an I/O memory range multiple times, pass the same paddr, count, and prot values in each call to kernel_iomap.

If a driver does not intend to share access to I/O memory with any user applications, it must call kernel_iomap with PROT_KRW, which restricts access to the kernel only. If the driver calls kernel_iomap with PROT_KRW, a subsequent call to user_iomap or user_iomap_private for the same I/O memory will fail.
You cannot change a mapping with PROT_KRW by calling kernel_iomap with PROT_URW. You must first remove the mapping in order to change the protection. If only user applications will access I/O memory and the driver will never access the I/O memory, the driver only needs to call user_iomap or user_iomap_private; it does not need to call kernel_iomap.

**RETURN VALUES**

<>NULL Kernel virtual address.
NULL Error. Mapping failed.

**CONSTRAINTS**

Do not call in interrupt context.
Do not call while holding a spinlock.

**EXAMPLES**

```c
void  *physaddr;
void  *useraddr;
volatile int   *iop;
size_t pagesize;
int   ioreg;

/*
 * Get the physical address as seen by the processor.
 * Verify that the physical address is pagesize aligned.
 */
physaddr = xyz;
pagesize = iomap_pagesize();
VASSERT(((uintptr_t)physaddr % pagesize) == 0);

/*
 * Memory map the kernel virtual address and set the
 * protection on the pages to also allow user access.
 */
iop = kernel_iomap(NULL, physaddr, 1L, PROT_URW);

if (iop == NULL) {
    /*
     * Failed to map kernel virtual address.
     * Add error handling code here.
     */
}

/*
 * Now memory map the user virtual address using the
 * same physical address and page count parameters.
 */
useraddr = user_iomap(NULL, physaddr, 1L);

if (useraddr == NULL) {
    /*
     * Failed to map user virtual address.
     * Add error handling code here.
     */
}

/*
 * Access an IO register in the memory mapped page.
 */
ioreg = *iop;
```
SEE ALSO

iomap_enable_wic(9F), iomap_pagesize(9F), kernel_iounmap(9F), user_iomap(9F), user_iomap_private(9F),
user_iounmap(9F).
NAME

kernel_iomap_public - Publicly map I/O memory to a kernel address.

SYNOPSIS

#include <sys/iomap.h>

void *kernel_iomap_public(
    void *vaddr,
    void *paddr,
    pgcnt_t count,
    prot_t prot
);

PARAMETERS

vaddr Virtual address requested. must be NULL.
paddr Physical address of the I/O memory location from the view of the processor.
count Count of pages to map.
prot Protection mode for the mapping: PROT_KRW or PROT_URW.

DESCRIPTION

The kernel_iomap_public function is a deprecated interface. The recommended interface is kernel_iomap.

Use of kernel_iomap_public is not advised because it creates an unprotected mapping for the device I/O memory. If mapped with kernel_iomap_public and PROT_URW, any 64-bit process that can form the mapped virtual address will be able to gain access to the I/O memory since the associated pages are mapped with public access protection (that is, with no protection key) and user privilege level access.

The kernel_iomap_public function memory maps into kernel address space a range of contiguous pages in I/O address space starting at the physical address paddr for count pages. The value returned is the kernel virtual address that maps onto the base of the I/O memory location specified by paddr and count.

The physical address is the 64-bit physical address from the view of the processor, and the address must be pagesize aligned. The pagesize can be obtained by calling iomap_pagesize.

The 64-bit physical address paddr is not necessarily the same address value that is obtained from the Base Address Register (BAR) of a PCI device. The caller may need to convert the BAR value to the appropriate 64-bit physical address value used by the processor.

The prot value specifies the protection mode: either PROT_KRW or PROT_URW. If set to PROT_KRW, protection on the memory mapped pages is set such that the kernel is granted access to the I/O memory location, but no user application may be allowed access.

If set to PROT_URW, the protection is set such that a user may be allowed to share access with the kernel. The user virtual address is obtained by calling user_iomap passing the same paddr and count parameters.

RETURN VALUES

<>NULL Kernel virtual address.
NULL Error. Mapping failed.

CONSTRAINTS

Do not call in an interrupt context.
Do not call while holding a spinlock.

EXAMPLES

```c
void  *physaddr;
void  *useraddr;
int   *iop;
sizet_t pagesize;
int   ioreg;

/*
 * Get the physical address as seen by the processor.
 * Verify that the physical address is pagesize aligned.
 */
physaddr = xyz;
pagesize = iomap_pagesize();
VASSERT(((uintptr_t)physaddr % pagesize) == 0);

/*
 * Publicly map the kernel virtual address and set the
 * protection on the pages to also allow user access.
 * WARNING: Public mapping allows all users, including
 * unauthorized and errant user applications, to access the I/O memory location.
 */
iop = kernel_iomap_public(NULL, physaddr, 1L, PROT_URW);

/*
 * Now memory map the user virtual address using the
 * same physical address and page count parameters.
 */
useraddr = user_iomap(NULL, physaddr, 1L);

/*
 * Access the I/O memory location.
 */
ioreg = *iop;

SEE ALSO
iomap_pagesize(9F), kernel_iomap(9F), kernel_iounmap(9F), user_i omap(9F), user_iomap_private(9F), user_iounmap(9F).
```
NAME

kernel_iounmap - Unmap I/O memory for the assigned kernel address.

SYNOPSIS

#include <sys/iomap.h>

int kernel_iounmap(
    void *vaddr,
    pgcnt_t count
);

PARAMETERS

vaddr Virtual address returned by kernel_iomap.

count Count of pages to unmap.

DESCRIPTION

The kernel_iounmap function removes the kernel virtual address assigned to map a range of
I/O memory pages.

The vaddr parameter is the virtual address returned by kernel_iomap, and the count parameters
must match the count parameter passed to kernel_iomap.

RETURN VALUES

0 Success.

1 Failure.

CONSTRAINTS

Do not call in an interrupt context.

Do not call while holding a spinlock.

EXAMPLES

void    *physaddr;
void    *kernaddr;
pgcnt_t count;
int     retval;

...

/*
 * Memory map the kernel virtual address onto the I/O memory
 * location for count number of pages.
 */
kernaddr = kernel_iomap(NULL, physaddr, count, PROT_KRW);

...

/*
 * Remove the mapping for the kernel virtual address.
 */
retval = kernel_iounmap(kernaddr, count);
VASSERT(retval == 0);

SEE ALSO

iomap_pagesize(9F), kernel_iomap(9F), user_iomap(9F), user_iomap_private(9F), user_iounmap(9F).
NAME

kfree - Deallocate kernel memory.

SYNOPSIS

#include <sys/malloc.h>

void kfree(
    void *addr,
    int type
);

PARAMETERS

addr Address of the kernel memory previously allocated by a call to kmalloc.
type The memory allocation pool type.

DESCRIPTION

The kfree kernel function deallocates (frees) kernel memory previously allocated by a call to kmalloc. The addr parameter must be the address returned by kmalloc and type must be the same type passed to kmalloc.

RETURN VALUES

None

CONSTRAINTS

None

SEE ALSO

kmalloc(9F)
NAME

kmalloc -- Allocate kernel memory.

SYNOPSIS

#include <sys/malloc.h>

void *kmalloc(
    size_t size,
    int type,
    arena_flags_t flags
);

PARAMETERS

size The number of bytes (size) of kernel memory to allocate.
type The memory allocation pool type.
flags Flag to indicate the caller cannot block and wait for kernel memory availability.

DESCRIPTION

The kmalloc kernel function allocates size bytes of kernel memory from the pool type specified. The memory allocated is always contiguous in the virtual address space, but might be discontiguous in the physical address space. If size is greater than PAGESIZE, the physical pages allocated might not be physically contiguous.

Drivers must allocate memory from one of the following pool types: M_IOSYS or M_DMA. The M_IOSYS pool is for general I/O purposes; the M_DMA pool is for DMA purposes. Independent Hardware Vendors (IHV) must specify the M_IHV pool for general I/O purposes of their drivers.

The flags parameter can optionally have the M_NOWAIT flag set. If M_NOWAIT is set and no memory is available from the requested pool type, the function returns NULL. Without M_NOWAIT set, the caller can be blocked and made to wait for memory to become available.

The M_NOWAIT flag must be set if kmalloc is either:
• Called in an interrupt context
• Called while holding a spinlock

If M_NOWAIT is set, the caller must be prepared to handle the case where no kernel memory has been allocated.

M_WAITOK must be passed as the flags parameter if the caller is able to block and wait for memory to become available.

RETURN VALUES

<>NULL Virtual address of kernel memory allocated.
NULL No kernel memory has been allocated. NULL is returned only when the M_NOWAIT flag is set, otherwise the caller is blocked and waits until memory becomes available.

CONSTRAINTS

If the M_NOWAIT flag is not set, the following conditions must be true:
• The caller must not be in an interrupt context
• No spinlocks are held

SEE ALSO

kfree(9F)
NOTES

It is generally a bad programming practice to allocate memory within critical sections of your code. The kernel memory allocator, including the `kmalloc` interface, might be required to do a lot of work to satisfy an allocation request, and will sleep to wait for memory to become available, if necessary. This can have a large negative impact on the performance of your code.

There are two critical sections in which sleeping is prohibited by the kernel:

- Interrupt routines (code running in interrupt context)
- Code executed while holding a spinlock.

The `M_NOWAIT` flag enables you to allocate memory in these contexts. If `M_NOWAIT` is specified, the kernel memory allocator only tries the fast paths of memory allocation, returning `NULL` if unsuccessful. In this case, your code must be prepared to deal with a memory allocation failure. Typically, this requires retrying the allocation outside the critical section with the `M_WAITOK` flag specified.

HP strongly recommends that whenever possible you structure your code so all memory allocations are done outside of critical sections.
NAME

kmem_arena_alloc - Allocates a fixed size memory object from the arena.

SYNOPSIS

#include <sys/vm_arena_iface.h>

void *kmem_arena_alloc(
    kmem_handle_t handle,
    arena_flags_t flags
);

PARAMETERS

handle   Opaque arena handle returned by kmem_arena_create.
flags    Allocation flags.

DESCRIPTION

The kmem_arena_alloc function allocates a memory object from the specified arena. This function is called for arenas created with fixed sized allocations.

The handle parameter is the value returned by kmem_arena_create. It represents an arena created with fixed sized memory objects.

The flags parameter may contain one of the following values:

M_WAITOK   Allow the thread to block and wait for memory allocation.
M_NOWAIT   Do not allow the thread to block and wait. The function returns NULL if memory allocation will block the thread (for example, put the thread to sleep).

RETURN VALUES

<> NULL   Success: Pointer to the memory object allocated.
NULL     Failure: Unable to allocate a memory object.

CONSTRAINTS

Do not call while holding a spinlock.

Do not call in an interrupt context if flags is set to M_WAITOK.

EXAMPLES

The following example allocates a fixed sized memory object from the arena.

kmem_arena_handle_t my_arena;
void *objp;

    /* Create a fixed sized arena with default attributes. */
    my_arena = kmem_arena_create(sizeof(my_type), "MY_ARENA_NAME", NULL, M_WAITOK);
    /*
    * Allocate a fixed sized memory object from the arena.
    */
    objp = kmem_arena_alloc(my_arena, M_NOWAIT);
    /*
    * Allocation can fail with M_NOWAIT, so check the return value.
    */
    if (objp == NULL) {
        /*
        * Put error handling code here.
        */
SEE ALSO

kmem_arena_create(9F), kmem_arena_destroy(9F), kmem_arena_free(9F), kmem_arena_init(9F),
kmem_arena_varalloc(9F)

NOTES

It is generally a bad programming practice to allocate memory within critical sections of your code. The kernel memory allocator, including the kmem_arena_alloc interface, might be required to do a lot of work to satisfy an allocation request, and will sleep to wait for memory to become available, if necessary. This can have a large negative impact on the performance of your code.

There are two critical sections in which sleeping is prohibited by the kernel:

- Interrupt routines (code running in interrupt context)
- Code executed while holding a spinlock.

The M_NOWAIT flag enables you to allocate memory in these contexts. If M_NOWAIT is specified, the kernel memory allocator only tries the fast paths of memory allocation, returning NULL if unsuccessful. In this case, your code must be prepared to deal with a memory allocation failure. Typically, this requires retrying the allocation outside the critical section with the M_WAITOK flag specified.

HP strongly recommends that whenever possible you structure your code so all memory allocations are done outside of critical sections.
**NAME**

kmem_arena_attr_init - Initializes the arena attributes in the kmem_arena_attr_t structure.

**SYNOPSIS**

```c
#include <sys/vm_arena_iface.h>

void kmem_arena_attr_init(
    kmem_arena_attr_t *kattr,
    size_t size
);
```

**PARAMETERS**

- `kattr`    Pointer to the kmem_arena_attr_t data structure.
- `size`     Size in bytes of the kmem_arena_attr_t data structure.

**DESCRIPTION**

The kmem_arena_attr_init function initializes the kmem_arena_attr_t structure that is passed to kmem_arena_create. The function offloads from the caller the burden of initializing all the elements in the structure. Normally, only a few elements, if any, need to be modified by the caller.

The `size` parameter specifies the size of the kmem_arena_attr_t structure. This allows binary compatibility with future extensions to the kmem_arena_attr_t structure. If new attributes are added, the caller is not required to recompile. The `size` parameter must be passed as sizeof(kmem_arena_attr_t).

The elements of the kmem_arena_attr_t structure are defined as follows:

- `kat_struct_size` Specifies the size of the structure passed by the caller to kmem_arena_attr_init. It enables future extensions to the kmem_arena_attr_t structure without breaking binary compatibility with the caller. Do not modify this element.

- `kat_ctor()` Pointer to the constructor function, `ctor`, which does the work of initializing memory objects added to refill the arena. The constructor is intended to implement object caching.

- `kat_dtor()` Pointer to the destructor function, `dtor`, which does the work of dismantling memory objects reclaimed from the arena. The destructor is intended to implement object caching.

- `kat_maxcnt` Specifies the maximum number of memory objects allocated for the arena. This is used only for fixed size memory objects.

- `kat_minfcnt` Specifies the minimum number of memory objects per SPU to be kept on the free list of the arena. This is used as an advisory for managing small objects.

- `kat_maxpgcnt` Specifies the maximum number of 4K pages allocated for the arena. This is used only for variable sized memory objects.

- `kat_refillcnt` Specifies the number of objects to be added to the free list per refill.

- `kat_flags` Specifies special attribute flags, which are defined as follows:
  - KAT_NO_LGPG The memory objects allocated from the arena will not use large (super) pages. Allocations will only use 4K pages, which is
KAT_ALLOC32
The memory objects allocated from the arena will have a 32-bit virtual address. This attribute is applicable only for PA-RISC implementations and is not recommended for general use.

KAT_CACHE_XLARGE_OBJECTS
The arena caches xlarge memory objects. By default, the arena will not cache xlarge objects to avoid memory fragmentation, but some modules that use xlarge objects may want to cache them to improve performance.

KAT_ALIGN_ON_SIZE
The memory objects allocated from the arena will be aligned on a power-of-two address. For example, a size of 200 bytes will be aligned on a 256 byte boundary. Similarly, a size of 6000 bytes will be aligned on 8K boundary.

KAT_ALIGN_ON_SIZE_COMPAT
This is to provide similar alignment as the deprecated MALLOC interface. Sizes that are less than 4K are aligned on size but sizes greater than or equal to 4K are aligned on 4K. For example, a size of 200 bytes will be aligned on a 256 byte boundary but a size of 6000 bytes is aligned on 4K.

KAT_CONTIGUOUS_PHYSMEM
The memory objects allocated from the arena will be contiguous in physical memory.

KAT.MULTICACHE_SIZE
The memory objects allocated from the arena will be a multiple of cacheline size. This is required by certain drivers for DMA transactions.

KAT_DMA32
The memory objects allocated from the arena will have a 32-bit physical address. This attribute is applicable only for IA64 implementations. It is required by drivers that control devices limited to 32-bit DMA addressing.

kat_align
Specifies the alignment of the object in the arena. It is used only for fixed size memory objects. The alignment for small objects must be a multiple of eight bytes and be a power of two. The alignment of objects greater than or equal to 4K are by default 4K-aligned and cannot be changed.
RETURN VALUES
None

CONSTRAINTS
May be called in user or interrupt context.
May be called while holding a spinlock.

EXAMPLES
The following example creates a variable sized arena.

```c
kmem_arena_attr_t attr;
kmem_arena_handle_t my_arena;

/*
 * Initialize the arena attributes to default values.
 */
kmem_arena_attr_init(&attr, sizeof(kmem_arena_attr_t));

/*
 * Alter the default arena attributes to align memory objects
 * on a power-of-two address. The power-of-two value is greater
 * than or equal to the size of the memory object allocated.
 */
attr.kat_flags |= KAT_ALIGN_ON_SIZE;

/*
 * Create my variable sized arena.
 */

my_arena = kmem_arena_create(0, "MY_ARENA_NAME", &attr, M_WAITOK);
```

SEE ALSO
kmem_arena_alloc(9F), kmem_arena_create(9F), kmem_arena_destroy(9F), kmem_arena_free(9F), kmem_arena_varalloc(9F).
NAME

kmem_arena_create - Creates and initializes an arena.

SYNOPSIS

#include <sys/vm_arena_iface.h>

void *kmem_arena_create(
    size_t size,
    char *arena_name,
    kmem_arena_attr_t *kattr,
    arena_flags_t flags
);

PARAMETERS

size Size in bytes of the memory objects allocated from the arena.
arena_name Name of the arena, which must be unique across the system.
kattr Pointer to the kmem_arena_attr_t structure.
flags Creation flags.

DESCRIPTION

The kmem_arena_create function creates and initializes an arena, which is a user defined pool of memory objects. Separate arenas must be created for unrelated memory objects that are frequently allocated. A single arena may be created for all infrequently allocated memory objects with similar attributes.

The size parameter specifies the size of memory objects allocated from the arena. If size is the value zero, the arena will allocate variable sized memory objects; otherwise, the arena will allocate memory objects of a fixed size. Variable sized memory objects are allocated by calling kmem_arena_varalloc and fixed sized memory objects are allocated by calling kmem_arena_alloc.

The arena_name parameter points to a zero-terminated character string that identifies the arena by name. Choose a descriptive name that will be unique across the system.

Attributes that describe the arena are passed through the kattr parameter. If this parameter is NULL, default attributes are applied to the arena. See kmem_arena_attr_init(9F) for information on how to initialize arena attributes.

The flags parameter may contain one of the following flag values:
M_WAITOK Allow the thread to block and wait while creating the arena.
M_NOWAIT Do not allow the thread to block and wait. The function returns NULL if creation of the arena will block the thread (for example, put the thread to sleep).

If successful, the function returns an opaque arena handle, which is used for all allocations and de-allocations of memory objects from and to the arena.

RETURN VALUES

<> NULL Success: Opaque arena handle.
NULL Failure: No arena allocated.

CONSTRAINTS

Do not call while holding a spinlock.
Do not call in interrupt context if flags is set to M_WAITOK.
EXAMPLES

The following example creates an arena with fixed sized memory objects and default attributes.

```c
kmem_arena_attr_t attr;
kmem_handle_t var_arena;
kmem_handle_t fixed_arena;

/*
 * Specify memory objects are to be aligned on a cacheline.
 */
kmem_arena_attr_init(&attr, sizeof(kmem_arena_attr_t));
attr.kat_flags |= KAT_MULTICACHE_SIZE;

/*
 * Create an arena with variable sized memory objects, where each
 * memory object is physically aligned on a cacheline.
 */
var_arena = kmem_arena_create(0,
   "MY_VARIABLE_ARENA_NAME", &attr, M_WAITOK);

/*
 * Create an arena with fixed sized memory objects and default attributes.
 */
fixed_arena = kmem_arena_create(sizeof(my_type),
   "MY_FIXED_ARENA_NAME", NULL, M_WAITOK);
```

SEE ALSO

`kmem_arena_alloc(9F), kmem_arena_destroy(9F), kmem_arena_free(9F), kmem_arena_init(9F), kmem_arena_varalloc(9F)`. 
NAME

kmem_arena_destroy - Destroys the arena.

SYNOPSIS

#include <sys/vm_arena_iface.h>

kmem_handle_t kmem_arena_destroy(
    kmem_handle_t handle
);

PARAMETERS

handle Opaque arena handle returned by kmem_arena_create.

DESCRIPTION

The kmem_arena_destroy function destroys the arena created by kmem_arena_create. All memory objects allocated from the arena must be freed before calling this function, and subsequent requests to allocate memory objects in the arena will fail.

RETURN VALUES

NULL This function always returns NULL. The caller can use this routine to reset the handle to the arena just freed.

CONSTRAINTS

Can be called in user or interrupt context.

Do not call while holding a spinlock.

EXAMPLES

The following example destroys the arena once it is no longer needed.

kmem_arena_handle_t my_arena;
    /* Create a variable sized arena with default attributes. */
    my_arena = kmem_arena_create(0, "MY_ARENA_NAME", NULL, M_WAITOK);
    ...
    /* * Destroy the arena now, as it is no longer needed. */
    my_arena = kmem_arena_destroy(my_arena);

SEE ALSO

kmem_arena_alloc(9F), kmem_arena_create(9F), kmem_arena_free(9F), kmem_arena_init(9F), kmem_arena_varalloc(9F)
NAME

kmem_arena_free - Frees a memory object to its associated arena.

SYNOPSIS

#include <sys/vm_arena_iface.h>

void *kmem_arena_free(
    void *objp,
    arena_flags_t flags
);

PARAMETERS

objp Pointer to the memory object to be freed.
flags Free flags.

DESCRIPTION

The kmem_arena_free function frees a memory object and returns it to its associated arena. Once the memory object is freed, it cannot be accessed. The memory object may have been allocated by either kmem_arena_alloc or kmem_arena_varalloc.

The flags parameter may contain one of the following flag values:
M_WAITOK Allow the thread to block and wait while freeing the memory object.
M_NOWAIT Do not allow the thread to block and wait.

RETURN VALUES

NULL This function always returns NULL. It can be used by the caller to reset the pointer to the memory object just freed.

CONSTRAINTS

Do not call while holding a spinlock.
Do not call in an interrupt context if flags set to M_WAITOK.

EXAMPLES

The following example frees the memory object that is no longer used.

kmem_arena_handle_t my_arena;
void *objp;

    /*
     * Create a variable sized arena with default attributes.
     */
    my_arena = kmem_arena_create(0, "MY_ARENA_NAME", NULL, M_WAITOK);
    ...
    /*
     * Allocate a variable sized memory object from the arena.
     */
    objp = kmem_arena_varalloc(my_arena, sizeof(my_type), M_NOWAIT);
    ...
    /*
     * Free the memory object now that we are done using it.
     */
    objp = kmem_arena_free(objp, M_NOWAIT);
SEE ALSO

kmem_arena_alloc(9F), kmem_arena_create(9F), kmem_arena_destroy(9F), kmem_arena_init(9F), kmem_arena_varalloc(9F)
**NAME**

kmem_arena_varalloc - Allocate a variable sized memory object from the arena.

**SYNOPSIS**

```
#include <sys/vm_arena_iface.h>

void *kmem_arena_varalloc(
    kmem_handle_t handle,
    size_t size,
    arena_flags_t flags
);
```

**PARAMETERS**

- **handle**: Opaque arena handle returned by `kmem_arena_create`.
- **size**: Size in bytes of requested memory object.
- **flags**: Allocation flags.

**DESCRIPTION**

The `kmem_arena_varalloc` function allocates a memory object from the specified arena. This function is called for arenas created with variable sized allocations.

The `handle` parameter is the value returned by `kmem_arena_create`. It represents an arena created with variable sized memory objects.

The `size` parameter is the requested size in bytes of the memory object to be allocated.

The `flags` parameter may contain one of the following flag values:

- **M_WAITOK**: Allow the thread to block and wait for memory allocation.
- **M_NOWAIT**: Do not allow the thread to block and wait. The function returns NULL if memory allocation will block the thread (for example, put the thread to sleep).

**RETURN VALUES**

- <> NULL: Success: Pointer to the memory object allocated.
- NULL: Failure: Unable to allocate a memory object.

**CONSTRAINTS**

Do not call while holding a spinlock.

Do not call in interrupt context if `flags` is set to M_WAITOK.

**EXAMPLES**

The following example allocates a variable sized memory object from the arena.

```c
kmem_arena_handle_t my_arena;
void  *objp;

/*
  * Create a variable sized arena with default attributes.
  */
my_arena = kmem_arena_create(0, "MY_ARENA_NAME", NULL, M_WAITOK);

  /*
  * Allocate a variable sized memory object from the arena.
  */
objp = kmem_arena_varalloc(my_arena, sizeof(my_type), M_NOWAIT);
/*
```
if (objp == NULL) {
    /*
    * Put error handling code here.
    */
}

SEE ALSO

kmem_arena_alloc(9F), kmem_arena_create(9F), kmem_arena_destroy(9F), kmem_arena_free(9F),
kmem_arena_init(9F)

NOTES

It is generally a bad programming practice to allocate memory within critical sections of your
code. The kernel memory allocator, including the kmem_arena_varalloc interface, might be
required to do a lot of work to satisfy an allocation request, and will sleep to wait for memory
to become available, if necessary. This can have a large negative impact on the performance of
your code.

There are two critical sections in which sleeping is prohibited by the kernel:
• Interrupt routines (code running in interrupt context)
• Code executed while holding a spinlock.

The M_NOWAIT flag enables you to allocate memory in these contexts. If M_NOWAIT is specified,
the kernel memory allocator only tries the fast paths of memory allocation, returning NULL if
unsuccessful. In this case, your code must be prepared to deal with a memory allocation failure.
Typically, this requires retrying the allocation outside the critical section with the M_WAITOK
flag specified.

HP strongly recommends that whenever possible you structure your code so all memory
allocations are done outside of critical sections.
**NAME**

kr_close_node - Close a registry key and free all resources.

**SYNOPSIS**

```c
#include <sys/krs.h>
int kr_close_node(
    kr_key_t key
);
```

**PARAMETERS**

*key* The KRS key representing the node that is to be closed.

**DESCRIPTION**

The `kr_close_node` routine closes a node, previously opened by a call to `kr_open_node` or `kr_link_node`. Upon successful completion, all resources associated with the open instance are freed; the key is no longer valid.

**RETURN VALUES**

- **KR_ERR_BADKEY** The specified *key* does not exist.
- **KR_SUCCESS** Request completed successfully.

**CONSTRAINTS**

Before using this routine, you must first acquire the key by making a call to `kr_open_node`.

**EXAMPLES**

```c
int kr_open_node(kr_key_t, char *, kr_flags_t, kr_key_t*);
int kr_close_node(kr_key_t);myfunc()
{
    int rv:
    kr_key_t mykey;
    rv = kr_open_node(KR_NOKEY, "/mynode", KR_CREATE, &mykey);
    if (rv != KR_SUCCESS) {
        return ERROR;
    }
    ...
    kr_close_node(mykey);
}
```

**SEE ALSO**

`kr_delete_node(9F)`, `kr_link_node(9F)`, `kr_open_node(9F)`. 
kr_delete_node(9F)

NAME

kr_delete_node - Delete the given registry tree node or branch.

SYNOPSIS

int kr_delete_node(
    kr_key_t key,
    kr_flags_t flags
);

PARAMETERS

key The KRS key representing the node or branch to be deleted.
flags The flags parameter values are as follows:
    KR_ALL If the target node is not a leaf node, delete the entire branch.
    KR_LINKS If any of the deleted nodes have links, delete all of the links associated with them.

DESCRIPTION

The kr_delete_node routine causes one or more nodes to be deleted from the KRS tree. If the KR_ALL flag is specified, the entire subtree starting at the node is deleted. If the KR_ALL flag is not specified, and the node has children, and there are no links to the node, the request fails. If the KR_LINKS flag is specified, all links to any of the deleted nodes are also deleted.

Deleting a node does not close the key. This is because a key to an open node can be shared between multiple threads of execution. Because there is no mechanism for identifying these threads to notify them of the deletion, the node must remain open until an explicit kr_close_node is performed. If a thread tries to use the key after it is deleted, but before it has been closed, the function returns KR_ERR_DELETED to inform it of the current state. After the node has been closed, the function returns KR_ERR_BADKEY; the key is no longer recognized by the system.

RETURN VALUES

KR_ERR_BADKEY The specified key does not exist.
KR_ERR_DELETED The node represented by the given KRS key has been deleted since the node was opened.
KR_ERR_NOTLEAF Target node is not a leaf node (if KR_ALL is not specified).
KR_ERR_PARAM Attempt to delete the root node.
KR_ERR_RDONLY An attempt was made from user-space to delete a read-only node or the child of a read-only node.
KR_SUCCESS Request completed successfully.

CONSTRAINTS

Before using this routine, you must first acquire the key by making a call to kr_open_node.

EXAMPLES

int kr_delete_node(kr_key_t, kr_flags_t);myfunc()
{
    kr_key_t mykey; /* Error checking omitted */
    kr_open_node(KR_NOKEY, "/mynode", KR_NOFLAGS, &mykey);
    kr_delete_node(mykey, KR_NOFLAGS);
    kr_close_node(mykey);
}
SEE ALSO

kr_close_node(9F), kr_link_node(9F), kr_open_node(9F).
kr_delete_value(9F)

NAME

kr_delete_value - Delete the specified named value from an open node.

SYNOPSIS

#include <sys/krs.h>

int kr_delete_value(
    kr_key_t key,
    char *vname,
    kr_flags_t flags
);

PARAMETERS

key The KRS key representing the node from which the value is to be deleted.
vname The name of the value to delete.
flags The valid value is KR_NOFLAGS.

DESCRIPTION

The kr_delete_value routine deletes the specified value from the node identified by key.

RETURN VALUES

KR_ERR_BADKEY The specified key does not exist.
KR_ERR_DELETED The node represented by the given KRS key has been deleted since the node was opened.
KR_ERR_NAME The length of the specified value name exceeds KR_NAME_LEN.
KR_ERR_NOTFOUND The specified value could not be found.
KR_ERR_RDONLY An attempt was made from user-space to delete a read-only value or delete a value in a read-only node.
KR_SUCCESS Request completed successfully.

CONSTRAINTS

Before using this routine, you must first acquire the key by making a call to kr_open_node.

EXAMPLES

int kr_delete_value(kr_key_t, char *, kr_flags_t);

myfunc()
{
    /* Error checking omitted */
    kr_open_node(KR_NOKEY, "/mynode", KR_NOFLAGS, &mykey);
    kr_delete_value(mykey, "v0", KR_NOFLAGS);
    kr_close_node(mykey);
}

SEE ALSO

kr_get_value(9F), kr_open_node(9F), kr_set_value(9F).
NAME

kr_flush - Cause KRS data to be saved to persistent storage.

SYNOPSIS

#include <sys/krs.h>

int kr_flush(void);

PARAMETERS

None

DESCRIPTION

Normally, changes made to persistent data in KRS are saved to disk at predetermined time intervals. If you call this routine, the system saves the data to disk as soon as possible.

The kr_flush routine sends a signal to the system daemon responsible for saving KRS data and returns as soon as the signal is sent; it does not wait for the data to be written to disk (non-blocking). Use the kr_get_mod_time routine to determine when the data is actually saved to disk.

To maximize performance efficiency, do not call kr_flush until you are done changing persistent data.

RETURN VALUES

KR_FAIL The KRS daemon was not running and the signal could not be sent. This must be a temporary situation as the daemon is respawned by init. If the signal is not being sent the daemon saves the data as soon as it is respawned.

KR_SUCCESS The signal to the daemon was sent successfully.

EXAMPLES

int kr_flush();

myfunc()
{
    kr_key_t   key;
    /* Error checking omitted */
    kr_open_node(KR_NOKEY, "/mynode", KR_CREATE |
                 KR_PERSISTENT, &key);
    kr_close_node(key);
    kr_flush();
}

SEE ALSO

kr_get_mod_time(9F), kr_open_node(9F).
NAME

kr_get_mod_time - Return the last time persistent data was modified in KRS.

SYNOPSIS

#include <sys/krs.h>

int kr_get_mod_time(
    kr_flag_field_t class,
    kr_flag_field_t phase,
    uint64_t *mtime
);

PARAMETERS

class The specific class of persistent data that is relevant to the request. The valid values are as follows:

KR_CLASS_KERNEL Return the last time kernel-specific persistent data was changed (governed by phase).
KR_CLASS_SYSTEM Return the last time system-specific persistent data was changed (governed by phase).
KR_CLASS_SYNC Return the last time KRS data was flushed to persistent storage. When specified, phase is ignored.

phase The phase within the appropriate class. This must be KR_PHASE1 or KR_PHASE_ANY.

mtime A pointer to caller-supplied storage, through which the modification time is returned.

DESCRIPTION

The kr_get_mod_time routine returns a time stamp indicating the last time persistent data matching the requested class and phase was modified. Use the returned time stamps only for relative comparisons. If time stamp B is greater than time stamp A, time stamp B indicates a more recent change.

RETURN VALUES

KR_ERR_PARAM Either an unrecognized class or unrecognized phase was specified.
KR_SUCCESS The request completed successfully.

CONSTRAINTS

None

EXAMPLES

int kr_get_mod_time(kr_flag_field_t, kr_flag_field_t, uint64_t);
int kr_get_file_name(char *, kr_size_t, kr_flag_field_t, kr_flag_field_t);
void kr_print_tree(kr_key_t, char *, int, int);

myfunc()
{
    /* Error checking and local declarations omitted */
    kr_get_mod_time(KR_CLASS_ANY, KR_PHASE_ANY, &time);
    kr_get_file_name(name_buf, &buf_sz, KR_FILE_SYS,
                     KR_PHASE1);
    kr_print_tree(KR_NOKEY, "/", 0, 0);
}
SEE ALSO

`kr_flush(9F), kr_open_node(9F).`
kr_get_node_info(9F)

NAME

kr_get_node_info - Get information about a given node.

SYNOPSIS

#include <sys/krs.h>

int kr_get_node_info(
    kr_key_t key,
    kr_flag_field_t *info_buf,
    kr_flag_field_t *vtyp_buf,
    kr_flag_field_t vclass,
    kr_flag_field_t vphase,
    kr_flag_field_t *flag_buf,
    kr_linkid_t *lid_buf
);

PARAMETERS

key The KRS key representing the node whose information is to be obtained.

info_buf A pointer to client-supplied storage through which informational flags are returned. These flags convey the following:

- KR_HASCHILDREN The node in question is not a leaf node. It has children.
- KR_LINKED The data associated with the node in question is linked to by at least one other node.

vtyp_buf A pointer to client-supplied storage through which bit fields indicating the types of values contained within the node are returned.

vclass Value class. When collecting information about the node's values, only include information about the values that match vclass, as follows:

- KR_CLASS_NONE Do not return any value information for the node.
- KR_CLASS_KERNEL Only return information for values that are flagged kernel-specific.
- KR_CLASS_SYSTEM Only return information for values that are flagged system-specific.
- KR_CLASS_ANY Return information for both kernel- and system-specific values.

vphase Value phase. When collecting information about the node's values, only include information about values matching vphase, as follows:

- KR_PHASE1 Match Phase 1 persistent values.
- KR_PHASE_ANY Match any persistent value.

flag_buf A pointer to client-supplied storage through which the attribute flags associated with the node in question are returned.

lid_buf A pointer to client-supplied storage through which the link ID associated with the node's data is returned. This link ID uniquely identifies the data portion of the node and is the same for all nodes linked to the data.

DESCRIPTION

The kr_get_node_info routine returns information specific to the given node. The information returned is as described in the Parameters section.
RETURN VALUES

KR_ERR_BADKEY The specified key does not exist.
KR_ERR_DELETED The node represented by the given KRS key has been deleted since the node was opened.
KR_SUCCESS Request completed successfully.

CONSTRAINTS

Before using this routine, you must first acquire the key by making a call to kr_open_node.

EXAMPLES

```c
int kr_get_node_info(kr_key_t, kr_flag_field_t, kr_flag_field_t,
    kr_flag_field_t, kr_link_id_t);
int kr_get_vinfo(kr_key_t, char *, kr_type_t *, kr_size_t *,
    kr_flag_field_t *);

myfunc()
{
    /* Error checking and local declarations omitted */
    kr_open_node(KR_NOKEY, "/mynode", KR_NOFLAGS, &key);
    kr_get_node_info(key, &info_buf, &vtype_buf, vclass,
        vphase, &flag_buf, &lid_buf);
    kr_get_vinfo(key, "v0", &type_buf, &size_buf, &vflag_buf);
    kr_close_node(key);
}
```

SEE ALSO

kr_get_node_names(9F), kr_open_node(9F), kr_set_node_flags(9F).
**NAME**

kr_get_node_names - Retrieve a list of node names, as specified by flags.

**SYNOPSIS**

```c
#include <sys/krs.h>

int kr_get_node_names(
    kr_key_t key,
    char *names[],
    int *n_name,
    char *buf,
    kr_size_t *buf_sz,
    kr_flags_t flags
);
```

**PARAMETERS**

- **key**
  The KRS key representing the node relative to which node names are to be retrieved.

- **names**
  A client-supplied array through which pointers to the node names are returned.

- **n_name**
  When called, the number of elements in the names array. On return, the actual number of names that have been returned or the required size of the names array if it is not large enough to fulfill the request.

- **buf**
  A client-supplied buffer through which the text of the node names are returned. Pointers returned in the names array point into this buffer.

- **buf_sz**
  When called, the size of the buffer pointed to by buf. On return, the actual number of characters returned through buf, or the required size of the buffer if it is not large enough to fulfill the request.

- **flags**
  The flags parameter values are as follows:

  - **KR_PATH**
    Return the names of all the nodes in the path to the target node.
  
  Otherwise, return the names of the target node’s children.

**DESCRIPTION**

The `kr_get_node_names` routine returns node names as governed by flags. If the KR_PATH flag is specified, the names of the nodes comprising the path to the target node are returned. Otherwise, the names of the node’s children are returned.

The names are returned through the names and buf parameters. The buf parameter points to a caller-supplied buffer that will contain the text of the names being returned. The names parameter points to a caller-supplied array of character pointers that will contain pointers to each name in buf. The n_name and buf_sz parameters point to caller-supplied storage containing the size of the names array and buf buffer respectively.

If the storage supplied by the caller is not enough to fulfill the request, the routine sets n_name and buf_sz to indicate the storage required; the request fails with KR_ERR_SIZE. Upon successful completion, n_name is set to the actual number of names returned.

When returning the names of the nodes comprising the node’s path, the names are listed from the target node to root. For example:

names[0] points to the name of the target node. names[n_name-1] points to the name of the root node.

**RETURN VALUES**

- **KR_ERR_BADKEY**
  The specified key does not exist.
KR_ERR_DELETED  The node represented by the given KRS key has been deleted since the node was opened.
KR_ERR_PARAM   If n_name or buf_sz is NULL, or *buf_sz is not zero and buf is NULL.
KR_ERR_SIZE    The size of names, or buf, or both names and buf, is not enough to fulfill the request.
KR_SUCCESS     Request completed successfully.

CONSTRAINTS
Before using this routine, you must first acquire the key by making a call to kr_open_node.

EXAMPLES
int kr_get_node_names(kr_key_t, char [*], int *, char *,
kr_size_t *, kr_flags_t);
int kr_get_value_names(kr_key_t, char [*], int *, char *,
kr_size_t *);
myfunc(kr_key_t key)
{
    /* Error checking and local declarations omitted */
    /* You can use the same call to determine the */
    /* array size or buffer size. */
    rv = kr_get_node_names(key, names, &num_names, buf, &buf_sz,
    KR_NOFLAGS);
    if (rv == KR_ERR_SIZE) { /* Array or buffer size is insufficient */
        names = malloc(sizeof(char *) * num_names);
        buf = malloc(buf_sz);
        /* You can use the same call to get the node names. */
        rv = kr_get_node_names(key, names, &num_names, buf, &buf_sz,
        KR_NOFLAGS);
    }
    for (i = 0; i < num_names; i++) {
        kr_open_node(key, names[i], KR_NOFLAGS, &ckey);
        ...
        myfunc(ckey);
        kr_close_node(ckey);
    }
}

SEE ALSO
kr_get_node_info(9F), kr_open_node(9F), kr_set_node_flags(9F).
NAME

kr_get_value - Get the specified named value from an open node.

SYNOPSIS

#include <sys/krs.h>

int kr_get_value(
    kr_key_t key,
    char *vname,
    kr_type_t *type_buf,
    kr_size_t *size_buf,
    void *buf,
    kr_flags_t flags
);

PARAMETERS

key The KRS key representing the node from which the value is to be read.

vname The name of the value to read.

type_buf When called, the expected value type. On return, the actual value type.

size_buf When called, the size of the client-supplied buffer, pointed to by buf. On return, the actual size of the value, or the required size of the buffer if it is not large enough to fulfill the request.

buf A pointer to the client-supplied buffer to which the value's data is to be copied.

flags The flags parameter values are as follows:

    KR_NO_INHERIT If the value to be read is not found directly in the target node, the routine does not check to see if the value is exported from any of the node's ancestors.

    KR_HOLD Used when reading values of type KR_VTYPE_TREEREF. Specifying this flag causes the hold count for the returned key to be incremented, preventing it from being freed while still in use. When the client is finished using the key, call kr_release_reference.

DESCRIPTION

The kr_get_value routine retrieves a value from an open node. The data of the value is copied into the client-supplied buffer pointed to by buf. The size_buf parameter points to a client-supplied variable containing the size of the buffer. If this size is not large enough to hold the data, the request will fail and the variable pointed to by size_buf will be set to the needed size. Upon successful completion, size_buf and type_buf are set to the actual size and type of value, respectively.

If the value is not found in the target node and the KR_NO_INHERIT flag is not specified, each of the target node's ancestors is searched for the value. The ancestor nodes are searched from the target node's parent toward the root node. If a value is found with the name vname and it has the KR_EXPORT flag set, it is returned to the caller.

The following value types are supported:

    KR_VTYPE_STRING A NULL terminated string.

    KR_VTYPE_USER User defined. Stored as uninterpreted data and size.


    KR_VTYPE_INT64 A 64-bit integer.
**RETURN VALUES**

- **KR_ERR_BADKEY**: The specified key does not exist.
- **KR_ERR_DELETED**: The node represented by the given KRS key has been deleted since the node was opened.
- **KR_ERR_NAME**: The length of the specified value name exceeds KR_NAME_LEN.
- **KR_ERR_NOTFOUND**: The specified value could not be found.
- **KR_ERR_PARAM**: If type_buf or size_buf are NULL, or if *size_buf is not zero and buf is NULL.
- **KR_ERR_SIZE**: The value is larger than the size of buf.
- **KR_SUCCESS**: Request completed successfully.

**CONSTRAINTS**

Before using this routine, you must first acquire the key by making a call to `kr_open_node`.

**EXAMPLES**

```c
int kr_get_value(kr_key_t, char *, kr_type_t *, kr_size_t *,
void *, kr_flags_t);
myfunc()
{
    int32_t type = KR_VTYPE_INT32;
    kr_type_t size = sizeof(int32_t);
    /* Error checking omitted */
    kr_open_node(KR_NOKEY, "mynode", KR_NOFLAGS, &mykey);
    kr_get_value(mykey, "v0", &type, &size, &value,
        KR_NOFLAGS);
    kr_close_node(mykey);
}
```

**SEE ALSO**

`kr_delete_value(9F)`, `kr_open_node(9F)`, `kr_release_reference(9F)`, `kr_set_value(9F)`.
NAME

kr_get_value_names - Retrieve a list of value names for the specified node.

SYNOPSIS

#include <sys/krs.h>

int kr_get_value_names(
    kr_key_t key,
    char *names[],
    int *n_name,
    char *buf,
    kr_size_t *buf_sz
);

PARAMETERS

key - The KRS key representing the node whose value names are to be read.

names - A client-supplied array through which pointers to the value names are returned.

n_name - When called, the number of elements in the names array. On return, the actual number of names that have been returned or the required size of the names array if it is not large enough to fulfill the request.

buf - A client-supplied buffer through which the text of the value names are returned. Pointers returned in the names array point into this buffer.

buf_sz - When called, the size of the buffer pointed to by buf. On return, the actual number of characters returned through buf, or the required size of the buffer if it is not large enough to fulfill the request.

DESCRIPTION

The kr_get_value_names routine returns the names of all the values in the given node. The names are returned through the names and buf parameters. The buf parameter points to a caller-supplied buffer that will contain the text of the names being returned. The names parameter points to a caller-supplied array of character pointers that will contain pointers to each name in buf. The n_name and buf_sz parameters point to caller-supplied storage containing the size of the names array and buf buffer respectively.

If the storage supplied by the caller is not enough to fulfill the request, n_name and buf_sz are set to indicate the storage required; the request fails with KR_ERR_SIZE. Upon successful completion, n_name is set to the actual number of names returned.

RETURN VALUES

KR_ERR_BADKEY - The specified key does not exist.

KR_ERR_DELETED - The node represented by the given KRS key has been deleted since the node was opened.

KR_ERR_PARAM - If n_name or buf_sz is NULL, or *buf_sz is not zero and buf is NULL.

KR_ERR_SIZE - The size of names, or buf, or both names and buf, is not enough to fulfill the request.

KR_SUCCESS - Request completed successfully.

CONSTRAINTS

Before using this routine, you must first acquire the key by making a call to kr_open_node.
EXAMPLES

int kr_get_node_names(kr_key_t, char *[], int *, char *,
    kr_size_t *, kr_flags_t);
int kr_get_value_names(kr_key_t, char *[], int *, char *,
    kr_size_t *);
myfunc(kr_key_t key)
{
    /* Error checking and local declarations omitted */
    kr_get_node_names(key, names, &num_names, buf, &buf_sz,
        KR_NOFLAGS);
    for (i = 0; i < num_names; i++) {
        kr_open_node(key, names[i], KR_NOFLAGS, &ckey);
        ...
        myfunc(ckey);
        kr_close_node(ckey);
    }

SEE ALSO

kr_get_vinfo(9F), kr_open_node(9F), kr_set_value_flags.
NAME

kr_get_vinfo - Get information about a given named value.

SYNOPSIS

#include <sys/krs.h>

int kr_get_vinfo(
    kr_key_t key,
    char *vname,
    kr_type_t *type_buf,
    kr_size_t *size_buf,
    kr_flag_field_t *flag_buf
);

PARAMETERS

key The KRS key representing the node containing the value whose information is to be obtained.
vname The name of the value whose information is to be obtained.
type_buf A pointer to client-supplied storage through which the value's type is returned.
size_buf A pointer to client-supplied storage through which the value's size is returned.
flag_buf A pointer to client-supplied storage through which the value's flags are returned.

DESCRIPTION

The kr_get_vinfo routine retrieves information specific to the value in question. If the value vname is found in the node associated with key, the value's size type and flags are returned to the caller.

RETURN VALUES

KR_ERR_BADKEY The specified key does not exist.
KR_ERR_DELETED The node represented by the given KRS key has been deleted since the node was opened.
KR_ERR_NAME The length of the specified value name exceeds KR_NAME_LEN.
KR_ERR_NOTFOUND The specified value could not be found.
KR_SUCCESS Request completed successfully.

CONSTRAINTS

Before using this routine, you must first acquire the key by making a call to kr_open_node.

EXAMPLES

int kr_get_node_info(kr_key_t, kr_flag_field_t, kr_flag_field_t,
    kr_flag_field_t, kr_flag_field_t,
    kr_flag_field_t, kr_link_id_t);
int kr_get_vinfo(kr_key_t, char *, kr_type_t *, kr_size_t *,
    kr_flag_field_t *);

myfunc()
{
    /* Error checking and local declarations omitted */
    kr_open_node(KR_NOKEY, "/mynode", KR_NOFLAGS, &key);
    kr_get_node_info(key, &info_buf, &vtype_buf, vclass,
        vphase, &flag_buf, &lid_buf);
    kr_get_vinfo(key, "v0", &type_buf, &size_buf, &vflag_buf);
kr_close_node(key);

SEE ALSO

kr_get_value_names(9F), kr_open_node(9F), kr_set_value_flags(9F).
**NAME**

kr_link_node - Create a new registry node as a link to an existing node.

**SYNOPSIS**

```c
#include <sys/krs.h>

int kr_link_node(
    kr_key_t skey,
    kr_key_t root_key,
    char *path,
    kr_flags_t flags,
    kr_key_t *new_key
);
```

**PARAMETERS**

- `skey`: The source key that represents the existing node to which to link.
- `root_key`: The KRS key representing the parent node for the first node defined in `path`. Here, `path` is relative to `root_key`.
- `path`: A string defining the path to the target node. Node names in the path string are separated by the '/' character. If `root_key` is NULL, the path is assumed to be absolute and must contain a leading '/' character. If `root_key` is not NULL, the path is assumed to be relative to `root_key` and must not contain a leading '/' character.
- `flags`: The flags passed through this parameter govern the action of the link operation and define the characteristics of the newly created node. The valid values are as follows:
  - **KR_ALL**: Create intermediate nodes in the path to the target node as needed.
  - **KR_NOWAIT**: It is not permissible to sleep while fulfilling the link request. Otherwise, the request could block.
  - **KR_NO_PATH_FLAGS**: When **KR_ALL** is used to create intermediate nodes, this flag forces intermediate nodes to have default flag settings. Otherwise, the flags are set to the value of the created target node.
  - **KR_PERSISTENT**: The newly created target node needs to be persistent across system reboot.
  - **KR_USER_RDONLY**: The new link, any existing links, and the target node are to be read-only from user-space.
  - **KR_PRUNE**: Do not save this node or any of its descendents to persistent storage. This effectively removes the branch in question on next reboot, even if information in the branch is flagged as persistent.
- `new_key`: A KRS key representing the new link node is returned through this parameter.

**DESCRIPTION**

The **kr_link_node** routine creates a new node that is a link to an existing node. The data and children of the original node are also recognized as the data and children of the new node. The original node is specified through the `skey` parameter, which contains the key associated with the node (as returned by a previous call to **kr_open_node**). The target node is specified through `root_key` and `path` (as in **kr_open_node**).
If the target node already exists, the request fails. If any other node in the path does not exist, the request fails, unless the KR_ALL flag was specified.

RETURN VALUES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KR_ERR_BADKEY</td>
<td>The specified skey or root_key does not exist.</td>
</tr>
<tr>
<td>KR_ERR_DELETED</td>
<td>The node represented by the given KRS key has been deleted since the node was opened.</td>
</tr>
<tr>
<td>KR_ERR_EXIST</td>
<td>The target node already exists.</td>
</tr>
<tr>
<td>KR_ERR_LOOP</td>
<td>The link would form a loop in the tree.</td>
</tr>
<tr>
<td>KR_ERR_NAME</td>
<td>A name component of the path string exceeds KR_NAME_LEN.</td>
</tr>
<tr>
<td>KR_ERR_NOMEN</td>
<td>When the KR_NOWAIT flag is specified, KRS cannot allocate enough memory to fulfill request.</td>
</tr>
<tr>
<td>KR_ERR_PARAM</td>
<td>Invalid parameter. (For example: new_key = NULL.)</td>
</tr>
<tr>
<td>KR_ERR_PATH</td>
<td>The string pointed to by path is not a valid path string.</td>
</tr>
<tr>
<td>KR_ERR_RDONLY</td>
<td>An attempt was made from user-space to create the link under a parent node that is read-only.</td>
</tr>
<tr>
<td>KR_ERR_TOODEEP</td>
<td>The depth of the target node exceeds KR_MAX_DEPTH.</td>
</tr>
<tr>
<td>KR_SUCCESS</td>
<td>Request completed successfully.</td>
</tr>
</tbody>
</table>

CONSTRAINTS

Before using this routine, you must first acquire the key by making a call to kr_open_node.

EXAMPLES

```c
int kr_link_node(kr_key_t, kr_key_t, char *, kr_flag_t, kr_key_t);
myfunc()
{
    kr_key_t           skey, key;

    /* Error checking omitted */
    kr_open_node(KR_NOKEY, "/mynode", KR_CREATE, &skey);
    kr_link_node(skey, KR_NOKEY, "/mynodelink", KR_NOFLAGS, &key);
    kr_close_node(key);
    kr_close_node(skey);
}
```

SEE ALSO

kr_close_node(9F), kr_open_node(9F).
NAME

kr_open_node - Open a given registry tree node for access.

SYNOPSIS

#include <sys/krs.h>

int kr_open_node(
    kr_key_t root_key,
    char *path,
    kr_flags_t flags,
    kr_key_t *new_key
);

PARAMETERS

root_key The KRS key representing the parent node for the first node defined in path. Here, path is relative to root_key.

path A string defining the path to the target node. Node names in the path string are separated by the '/' character. If root_key is NULL, the path is assumed to be absolute and must contain a leading '/' character. If root_key is not NULL, the path is assumed to be relative to root_key and must not contain a leading '/' character.

flags Specifies the action of the open operation and defines the characteristics of any nodes that are created. Valid values are as follows:

   KR_CREATE Create the node or nodes in question if they do not already exist.

   KR_ALL Create intermediate nodes in the path to the target node as needed. This must be used in conjunction with the KR_CREATE flag.

   KR_NOWAIT It is not permissible to sleep while fulfilling the request. Otherwise, the open request could block.

   KR_NO_PATH_FLAGS When KR_ALL is used to create intermediate nodes, this flag forces intermediate nodes to have default flag settings. Otherwise, the flags are set to the value of the created target node.

   KR_PERSISTENT Nodes created will be persistent across system reboot.

   KR_USER_RDONLY Any created nodes will be read only when accessed from user-level applications.

   KR_PRUNE Do not save this node or any of its descendents to persistent storage. This will effectively remove the branch in question on next reboot, even if information in the branch is flagged as persistent.

new_key A KRS key representing the newly opened node is returned through this parameter.

DESCRIPTION

The kr_open_node routine opens a specific node within the KRS tree, returning a key through which the node’s data can be accessed.

The node is identified by path, which can be absolute (starting from root), or relative to root_key (a key returned by a previous call to kr_open_node). The component ".." (dot dot) can be included in a path string to specify the parent of a given node.
If the target node does not exist, it is created if the KR_CREATE flag was specified. If any nodes other than the target node do not exist, they are created if both the KR_CREATE and KR_ALL flags were specified.

RETURN VALUES

KR_ERR_BADKEY The specified root_key does not exist.
KR_ERR_DELETED The node represented by the given KRS key has been deleted since the node was opened.
KR_ERR_NAME A name component of the path string exceeds KR_NAME_LEN.
KR_ERR_NOMEN When the KR_NOWAIT flag is specified KRS cannot allocate enough memory to fulfill request.
KR_ERR_NOTFOUND The target node does not exist and the KR_CREATE flag has not been specified.
KR_ERR_PARAM Invalid parameter. For example: new_key = NULL.
KR_ERR_PATH The string pointed to by path is not a valid path string.
KR_ERR_RDONLY An attempt was made from user-space to create a node under a parent that is read-only.
KR_ERR_TOODEEP The depth of the target node exceeds KR_MAX_DEPTH.
KR_SUCCESS Request completed successfully.

CONSTRAINTS

Before using this routine, you must first acquire the key by making a call to kr_open_node.

EXAMPLES

```c
int kr_open_node(kr_key_t, char *, kr_flags_t, kr_key_t*);
int kr_close_node(kr_key_t);myfunc()
{
    int rv;
    kr_key_t mykey;
    rv = kr_open_node(KR_NOKEY, "/mynode", KR_CREATE, &mykey);
    if (rv != KR_SUCCESS) {
        return ERROR;
    }
    ...
    kr_close_node(mykey);
}

SEE ALSO

kr_close_node(9F), kr_delete_node(9F), kr_link_node(9F), kr_release_reference(9F).```
kr_release_reference(9F)

NAME

kr_release_reference - Release a hold placed on a tree reference key by kr_get_value.

SYNOPSIS

#include <sys/krs.h>

int kr_release_reference(
    kr_key_t key
);

PARAMETERS

key The key to be released. This key must be a tree reference obtained through kr_get_value specifying the KR_HOLD flag.

DESCRIPTION

The kr_release_reference routine releases the hold that was placed on a tree reference key that was obtained through a call to kr_get_value. The key passed to this routine must be a tree reference obtained through kr_get_value specifying the KR_HOLD flag. Call this routine when you no longer need the key in question. Once released, do not be use the key without first reacquiring and holding it through kr_get_value.

RETURN VALUES

KR_ERR_BADKEY Indicates that the key in question does not exist, or that it was never held.
KR_SUCCESS The request completed successfully.

EXAMPLES

int kr_release_reference(kr_key_t);

myfunc()
{
    kr_key_t key, ref_key;
    kr_type_t type = KR_VTYPE_TREEREF;
    kr_size_t size = sizeof(kr_key_t);

    /* Error checking omitted */
    kr_open_node(KR_NOKEY, "/mynode", KR_NOFLAGS, &key);
    kr_get_value(key, "ref", &type, &size, &ref_key, KR_HOLD);
    kr_close_node(key);
    ...
    /* Use ref_key */
    kr_release_reference(ref_key);
}

SEE ALSO

kr_delete_node(9F), kr_get_value(9F), kr_link_node(9F), kr_open_node(9F).
NAME

kr_set_node_flags - Modify the set of flags associated with a given node.

SYNOPSIS

#include <sys/krs.h>

int kr_set_node_flags(
    kr_key_t key,
    kr_flags_t flags,
    kr_flags_field_t mode
);

PARAMETERS

key A KRS key representing the node whose flags are to be modified.
flags The flag values to be modified according to mode. Valid values are as follows:
    KR_LINKS Apply the flags to the node in question and all nodes that are
              linked to it.
    KR_PERSISTENT Nodes created will be persistent across system reboot.
    KR_USER_RDONLY Any created nodes will be read-only when accessed from
              user-level applications.
    KR_PRUNE Do not save this node or any of its descendents to persistent
              storage. This effectively removes the branch in question on next
              reboot, even if information in the branch is flagged as persistent.
mode If mode is KR_SET, the flag values in flags are set in the node. If mode is KR_CLEAR, the
      flag values in flags are cleared.

DESCRIPTION

Use the kr_set_node_flags routine to set or clear attribute flags in an existing node. If mode
is KR_SET, the flags specified in flags are set in the node. Otherwise, they are cleared.

RETURN VALUES

KR_ERR_BADKEY The specified key does not exist.
KR_ERR_PARAM The value of mode is not KR_SET or KR_CLEAR; or an attempt was made
    to set the KR_PRUNE flag on the root node.
KR_ERR_RDONLY An attempt was made from user-level to set the flags of a read-only node.
KR_SUCCESS The request completed successfully.

CONSTRAINTS

Before using this routine, you must first acquire the key by making a call to kr_open_node.

EXAMPLES

int kr_set_node_flags(kr_key_t, kr_flags_t, kr_flag_field_t);
int kr_set_value_flags(kr_key_t, char *, kr_flags_t,
    kr_flag_field_t);
myfunc()
{
    kr_key_t mykey1, mykey2;

    /* Error checking omitted */
    kr_open_node(KR_NOKEY, "/mynode", KR_NOFLAGS, &mykey1);
    kr_open_node(mykey1, "mychild",
KR_USER_RDONLY | KR_CREATE, &mykey2);
kr_set_node_flags(mykey1, KR_PERSISTENT1, KR_SET);
kr_set_value_flags(mykey1, "v0", KR_EXPORT, KR_CLEAR);
kr_close_node(mykey1);
kr_close_node(mykey2);

SEE ALSO

kr_get_node_info(9F), kr_get_node_names(9F), kr_open_node(9F).
NAME

kr_set_value - Set the specified named value in an open node.

SYNOPSIS

#include <sys/krs.h>

int kr_set_value(
    kr_key_t key,
    char *vname,
    kr_type_t type,
    kr_size_t size,
    void *buf,
    kr_flags_t flags
);

PARAMETERS

key     The KRS key representing the node in which the value is to be set.
vname   The name of the value being set.
type    The type of the value being set.
size    The size of the value being set.
buf     A pointer to the value's data.
flags   Flags governing the action of the set value operation and defining characteristics of
         the value being set. Valid values are as follows:
         KR_NOWAIT     It is not permissible to sleep while fulfilling the set value request.
                       Otherwise, the request might block.
         KR_PERSISTENT The value being set needs to be persistent across system reboot.
         KR_USRBUF     The data for the value being set is to be maintained in the buffer
                       supplied by the client. Otherwise, a new buffer will be allocated
                       by KRS.
         KR_EXPORT     The value being set is to be visible to all lower nodes in the
                       branch.
         KR_USER_RDONLY Any created values will be read-only when accessed from
                       user-level applications.

DESCRIPTION

The kr_set_value routine creates or changes a value in the specified node. The data for the
value in question is specified through buf, a pointer to storage containing the data in question.
Normally, KRS allocates storage to hold the data for the value, and the data is copied from
the user's buffer to the newly allocated storage. If the KR_USRBUF flag is specified however, the
buffer supplied by the caller is used instead. It is the caller's responsibility to ensure that the
buffer exists for the lifetime of the value.

The following value types are supported:

KR_VTYPE_STRING A NULL terminated string.
KR_VTYPE_USER   User-defined. Stored as un-interpreted data and size.
KR_VTYPE_INT64  A 64-bit integer.
KR_VTYPE_UINT64  An unsigned 64-bit integer.
KR_VTYPE_TREEREF Tree reference. Stored as a persistent KRS key.
KR_VTYPE_ADDR A memory address.

**RETURN VALUES**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KR_ERR_BADKEY</td>
<td>The specified <em>key</em> does not exist.</td>
</tr>
<tr>
<td>KR_ERR_DELETED</td>
<td>The node represented by the given KRS key has been deleted since the node was opened.</td>
</tr>
<tr>
<td>KR_ERR_INVAL_FLAG</td>
<td>Invalid flag specification.</td>
</tr>
<tr>
<td>KR_ERR_NAME</td>
<td>The length of the specified value name exceeds KR_NAME_LEN.</td>
</tr>
<tr>
<td>KR_ERR_NOMEN</td>
<td>When the KR_NOWAIT flag is specified, KRS cannot allocate enough memory to fulfill request.</td>
</tr>
<tr>
<td>KR_ERR_PARAM</td>
<td>Invalid parameter/flag combination.</td>
</tr>
<tr>
<td>KR_ERR_RDWRONLY</td>
<td>An attempt was made, from user-space, to change a read-only value, or change or create a value in a read-only node.</td>
</tr>
<tr>
<td>KR_ERR_TYPE</td>
<td>Unrecognized <em>type</em> specified.</td>
</tr>
<tr>
<td>KR_SUCCESS</td>
<td>Request completed successfully.</td>
</tr>
</tbody>
</table>

**CONSTRAINTS**

Before using this routine, you must first acquire the key by making a call to `kr_open_node`.

**EXAMPLES**

```c
int kr_set_value(kr_key_t, char *, kr_type_t, kr_size_t, void *, kr_flags_t);
myfunc()
{
    int32_t value = 123;
    kr_type_t type = KR_VTYPE_INT32;
    kr_size_t size = 0;

    /* Error checking omitted */
    kr_open_node(KR_NOKEY, "/mynode", KR_NOFLAGS, &mykey);
    kr_set_value(mykey, "v0", type, size, &value, KR_NOFLAGS);
    kr_close_node(mykey);
}
```

**SEE ALSO**

`kr_delete_value(9F), kr_get_value(9F), kr_open_node(9F)`.
kr_set_value_flags(9F)

NAME

kr_set_value_flags - Modify the set of flags associated with a given value.

SYNOPSIS

#include <sys/krs.h>

int kr_set_value_flags(
   kr_key_t key,
   char *vname,
   kr_flags_t flags,
   kr_flag_field_t mode
);

PARAMETERS

key A KRS key representing the node containing the value whose flags are to be modified.
vname The name of the value whose flags are to be modified.
flags The flag values to be modified according to mode. Valid values are as follows:
   KR_PERSISTENT The value to be set needs to be persistent across system reboot.
   KR_USRBUF The data for the value to be set is to be maintained in the buffer supplied by the client. Otherwise, KRS allocates a new buffer.
   KR_EXPORT The value being set is to be visible to all lower nodes in the branch.
   KR_USER_RDONLY Any created nodes will be read-only when accessed from user-level applications.
mode Determines how the flag values are modified. Valid mode values are as follows:
   KR_SET The flag values in flags are set in the value.
   KR_CLEAR The flag values in flags are cleared in the non-pending value.

DESCRIPTION

Use the kr_set_value_flags routine to set or clear attribute flags for an existing value. If mode is KR_SET, the flags specified in flags are set in the node. Otherwise, they are cleared.

RETURN VALUES

KR_ERR_BADKEY The specified key does not exist.
KR_ERR_NAME The length of the specified value name exceeds KR_NAME_LEN.
KR_ERR_NOTFOUND The specified value could not be found.
KR_ERR_PARAM The value of mode is not KR_SET or KR_CLEAR.
KR_ERR_RDONLY An attempt was made, from user-space, to set the flags of a read-only value or to set the flags of a value in a read-only node.
KR_SUCCESS The request completed successfully.

CONSTRAINTS

Before using this routine, you must first acquire the key by making a call to kr_open_node.

EXAMPLES

int kr_set_node_flags(kr_key_t, kr_flags_t, kr_flag_field_t);
int kr_set_value_flags(kr_key_t, char *, kr_flags_t, kr_flag_field_t);

150 Kernel Reference Pages
myfunc()
{
    kr_key_t mykey1, mykey2;

    /* Error checking omitted */
    kr_open_node(KR_NOKEY, "/mynode", KR_NOFLAGS, &mykey1);
    kr_open_node(mykey1, "mychild",
            KR_USER_RDCALLY | KR_CREATE, &mykey2);
    kr_set_node_flags(mykey1, KR_PERSISTENT1, KR_SET);
    kr_set_value_flags(mykey1, "v0", KR_EXPORT, KR_CLEAR);
    kr_close_node(mykey1);
    kr_close_node(mykey2);
}

SEE ALSO

kr_get_value_names(9F), kr_get_vinfo(9F), kr_open_node(9F).
kthread_tid_self(9F)

NAME

kthread_tid_self - Get the thread ID of the calling thread.

SYNOPSIS

#include <sys/types.h>
#include <sys/kthread_id.h>
tid_t kthread_tid_self(void);

PARAMETERS

None

DESCRIPTION

The kthread_tid_self obtains the thread ID of the calling thread.

RETURN VALUES

n  n is a nonnegative thread ID.

CONSTRAINTS

This routine cannot be called on the ICS.

EXAMPLES

#include <sys/types.h>
#include <sys/kthread_id.h>

    /*
     * Example : Get the tid of the calling thread
     */

    {
        tid_t tid;
        tid = kthread_tid_self();
    }

WARNINGS

None

SEE ALSO

None
NAME
kthreadp_self - Get a pointer to the kthread structure for calling thread

SYNOPSIS
#include <sys/kthread.h>
kthread_t *kthreadp_self(void);

DESCRIPTION
The kthread structure (struct kthread or kthread_t) is a core kernel private structure of indeterminate size.

RETURN VALUES
The kthreadp_self routine returns a pointer to the kthread structure associated with the calling thread.

CONSTRAINTS
None

SEE ALSO
procp_self(9F)
NAME
Ktimeout - Execute a callout function after a specified length of time at driver level interrupt priority.

SYNOPSIS
#include <sys/callout.h>
#include <sys/param.h>
callout_t *Ktimeout(
    int (*func)(),
    caddr_t arg,
    int t,
    void *dummy
);

PARAMETERS
func Function to execute when the time value t expires.
arg Argument passed to the callout function func.
t Time value in number of clock ticks.
dummy Must be NULL.

DESCRIPTION
The Ktimeout kernel function executes the specified callout function func after t clock ticks have expired. Execution of func takes place in an interrupt context at priority level 5 (driver level interrupt priority) where external interrupts to the processor are disabled. Drivers are encouraged to use the preferred interface timeout, where func is scheduled to execute at priority level 2.

The func parameter is a pointer to a function that takes one argument. Although the prototype declares the function to return an int value, the kernel does not make use of the return value.

The arg parameter is passed as the one argument to func.

The t parameter specifies the number of clock ticks to wait before calling func. To express time in seconds, multiply t by Hz, where Hz is defined as the number of clock ticks per second in <sys/param.h>.

The call to Ktimeout returns immediately without waiting for the time value t to expire. The timeout can be cancelled by making a corresponding call to untimeout.

RETURN VALUES
The Ktimeout function returns a pointer to a callout structure.

CONSTRAINTS
Do not call while holding a spinlock of order >= CALLOUT_LOCK_ORDER.

WARNINGS
Callout resources are not dynamically expandable. Each call to Ktimeout allocates a callout resource, and the resource is not released until the time value expires or the timeout is cancelled. The kernel may panic if no callout resources are available.

EXAMPLE
/*
 * Set a timeout to call my_timeout_func passing my_arg
 * after 5 seconds have expired.
 */
(void) Ktimeout(my_timeout_func, my_arg, 5*HZ, NULL);
...

static int
my_timeout_func(caddr_t arg)
{
    ...
}

SEE ALSO

timeout(9F), untimeout(9F)
NAME
ktune_canauto - Checks whether a tunable is capable of being automatically tuned.

SYNOPSIS
#include <sys/ktune.h>
int ktune_canauto(
    ktune_id_t tuneid
);

PARAMETERS
tuneid ID of the tunable to check.

DESCRIPTION
This function checks whether the tunable with the specified ID is capable of being automatically
tuned. A tunable is considered to be capable of being automatically tuned if it has one or more
registered handlers capable of computing an algorithmic default value, and has one or more
registered handlers capable of making dynamic changes to the tunable's value.

RETURN VALUES
<>0 Tunable is capable of being automatically tuned.
0 Tunable is not capable of being automatically tuned.

CONSTRAINTS
Do not call while holding a spinlock with order >= KTUNE_LISTLOCK_ORDER.

EXAMPLES
if (ktune_canauto(tuneid)) {
    ...
}

SEE ALSO
ktune_id(9F), ktune_isauto(9F)
NAME

ktune_current - Get the current value of a tunable from a tunable handler.

SYNOPSIS

#include <sys/ktune.h>

int ktune_current(
    ktune_id_t tuneid,
    uint64_t *value
);

PARAMETERS

tuneid    ID of the tunable of interest.
value     Pointer to a variable into which the current value of the tunable will be placed.

DESCRIPTION

This function retrieves the current value of a tunable. It is for use within tunable handlers only.
Other callers wanting the current value of a tunable must call ktune_get.
The value returned by this function ignores any tunable change transaction which might be in progress. To retrieve the pending value of a tunable change, call ktune_pending.

RETURN VALUES

0      Success
<>0    Error code returned from tunable handler

CONSTRAINTS

Call this function only from a tunable handler.

EXAMPLES

/* From within a tunable handler: */
ret = ktune_current(event_data->kte_tuneid, &currentval);
if (ret)
    ...

SEE ALSO

ktune_event_t(9F), ktune_get(9F), ktune_handler(9F), ktune_id(9F), ktune_pending(9F).
**NAME**

ktune_error - Log a tunable error message from a tunable handler.

**SYNOPSIS**

```c
#include <sys/ktune.h>
void ktune_error(
    ktune_txn_id_t txnid,
    const char *format,
    ...
);
```

**PARAMETERS**

- **txnid**: ID of the transaction in flight, if any.
- **format**: printf-style format string.
- **...**: Arguments needed for format string.

**DESCRIPTION**

This function logs a tunable change error message. If a tunable change transaction is in progress, the error message will be returned to the caller of `settune`. Otherwise, it will be logged to the system message buffer. This function must only be called from a tunable handler.

Do not embed tunable names in error message strings. Instead, put a `%s` where the name is to go, and use `ktune_name` as an argument to fill in the name.

**RETURN VALUES**

None

**CONSTRAINTS**

Call this function only from a tunable handler.

**EXAMPLES**

```c
/* From within a tunable handler: */
k tune_error(event_data->kte_txnid,
    "%lu is an invalid value for %s.",
    value, ktune_name(event_data->kte_tuneid));
```

**SEE ALSO**

`ktune_event_t(9F), ktune_handler(9F), ktune_id(9F), ktune_name(9F), ktune_warning(9F)`. 
NAME

ktune_event_t - Details of a tunable handler request.

SYNOPSIS

#include <sys/ktune.h>

typedef struct ktune_event {
    ktune_id_t kte_tuneid;
    ktune_txnid_t kte_txnid;
    ktune_op_t kte_op;
    ktune_event_flags_t kte_flags;
    ktune_id_t kte_baseid;
} ktune_event_t;

DESCRIPTION

A pointer to a ktune_event_t structure is passed as the event_data parameter in any invocation of a tunable handler that has a reason code of KEN_EVENT or KEN_BACKOUT. The data in this structure gives information about what the handler is expected to do.

STRUCTURE MEMBERS

kte_tuneid    Tunable ID of the tunable for which the handler is being called.
kte_txnid    Contains the transaction ID of the tunable change transaction in progress, if any. If no transaction is in progress, this value will be KTUNE_TXN_NULL.
kte_op       Specifies the operation that the handler is supposed to perform. Valid values are KTOP_CAPABLE, KTOP_GETDEFAULT, KTOP_VALIDATE, KTOP_PREPARE, KTOP_COMMIT, and KTOP_NOTIFY.
kte_flags    Additional information about the tunable and the transaction. Valid flags are:
             KTF_DEFAULT    Specifies that the tunable is in its default state.
             KTF_NEXTBOOT   Specifies that the transaction in progress will not take effect until boot.
kte_baseid   If the tunable for which the handler is being called is a derived tunable, this field contains the tunable ID of the corresponding base tunable; otherwise, this field contains KTUNE_ID_NULL.

SEE ALSO

ktune_handler(9F).
NAME

ktune_get - Activates a tunable and returns its value.

SYNOPSIS

#include <sys/ktune.h>

uint64_t ktune_get (const char *tunable, uint64_t failsafe);

PARAMETERS

tunable Name of the tunable to activate.
failsafe Value to use for the tunable if stored values are unavailable.

DESCRIPTION

This function activates a tunable. Tunables are inactive at boot time until this function is first called.

Before activating a tunable, all handlers for that tunable must be registered using ktune_register_handler.

If tunable failsafe mode was engaged at boot time (using the -tm boot flag), or if an error occurs trying to retrieve tunable values from the kernel registry, this function will return the failsafe value passed into it.

If the tunable is not in its default state, the saved value of the tunable is validated. If the saved value is no longer valid, a warning message is printed and the tunable is put into its default state.

If the tunable is in its default state, and a handler is registered that can compute default values, a new default value is calculated.

The resulting value of the tunable is returned.

This function may be used when the tunable is already active. In such cases it merely returns the current value of the tunable.

RETURN VALUES

Value of the tunable if it is found and the kernel is not in failsafe mode.

Failsafe value that is passed in otherwise.

CONSTRAINTS

Do not call while holding a spinlock.
Do not call in interrupt context.

EXAMPLE

maxuprc = ktune_get("maxuprc", MAXUPRCFAILSAFE);

SEE ALSO

ktune_register_handler(9F).
ktune_handler(9F)

NAME

ktune_handler - Tunable handler function.

SYNOPSIS

#include <sys/ktune.h>

int ktune_handler(
    ken_id_t eventid,
    int reason,
    ken_instance_t instance,
    void *handler_data,
    ktune_event_t *event_data,
    int *result
);

PARAMETERS

eventid   ID of the KEN event for the tunable.
reason    Reason the handler was invoked, one of:
          KEN_REGISTER  The handler is being registered.
          KEN_EVENT    The handler needs to perform a tunable operation.
          KEN_BACKOUT  The handler needs to reverse a tunable operation.
          KEN_UNREGISTER The handler is being unregistered.
instance   ID of this particular invocation of this handler.
handler_data A copy of the handler_data parameter provided to
              ktune_register_handler when the handler was registered.
event_data When the handler is invoked with reason set to KEN_EVENT or KEN_BACKOUT,
              this parameter points to a ktune_event_t structure describing the tunable
              operation to be performed or reversed. For other reason values, this parameter
              is NULL.
result     Points to an integer variable into which the handler must store its result code.

DESCRIPTION

A tunable handler is a function provided by a kernel subsystem that embodies knowledge of
how to manage kernel tunables owned by that subsystem.

A tunable handler can be registered for multiple tunables. Multiple tunable handlers can be
registered for a tunable. A tunable handler may be registered more than once for the same tunable
as long as the associated handler_data parameters differ. Tunable handlers are registered using
the ktune_register_handler call.

The actions taken by a handler depend on the reason and operation codes it receives. When the
reason code is KEN_EVENT or KEN_BACKOUT, the event_data parameter will point to a
ktune_event_t structure, which contains an operation code in its kte_op field (see
ktune_event_t(9F).) Valid operation codes are:

KTOP_CAPABLE  Return the capabilities of the handler.
KTOP_COMMIT   Commit the change to the value of the tunable.
KTOP_GETDEFAULT Compute a default value for the tunable.
KTOP_NOTIFY   Adjust to a new value of the tunable.
KTOP_PREPARE  Prepare to change the value of the tunable.
KTOP_VALIDATE Validate a proposed value for the tunable.

All handlers are required to respond to a reason/operation code combination of KEN_EVENT/KTOP_CAPABLE by setting their outbound result parameter to the bitwise OR of the KTOP_* constants for the operations implemented by the handler. Handling of all other reason and operation codes is optional.

When responding to a request, handlers take appropriate action for the request, set their outbound result parameter to a valid result code for the request, and return KEN_DONE. When handlers receive a request they do not implement, they must take no action other than returning KEN_DONE.

There are a variety of functions provided by the tunable infrastructure for use by handlers. See the SEE ALSO section.

RETURN VALUES

KEN_DONE Handler functions always return KEN_DONE.

CONSTRAINTS

Handler functions may be called during early boot or during a system call. They will not be called from the interrupt stack. They will not be called with any locks held, and must not hold any locks when they return.

Handlers that wish to sleep must first verify that the system has finished booting. Aside from that restriction, handlers may sleep.

EXAMPLES

#include <sys/ktune.h>

int template_tunable_handler(ken_id_t eventid,
int reason,
ken_instance_t instance,
void *handler_data,
ktune_event_t *event_data,
int *result)
{
    switch (reason) {
    case KEN_EVENT:
        switch (event_data->kte_op) {
        case KTOP_CAPABLE:
            *result = KTOP_CAPABLE | KTOP_... | ...;
            break;

        case KTOP_VALIDATE:
            ...
            break;
        case KTOP_GETDEFAULT:
            ...
            break;
        case KTOP_PREPARE:
            ...
            break;
        case KTOP_COMMIT:
            ...
            break;
        case KTOP_NOTIFY:
            ...
            break;
        default:
            break;
        }
    break;
case KEN_BACKOUT:
    switch (event_data->kte_op) {
    case KTOP_PREPARE:
        ...
        break;
    default:
        break;
    }
    break;

default:
    break;
}

return KEN_DONE;

SEE ALSO

ktune_canauto(9F), ktune_current(9F), ktune_error(9F), ktune_event_t(9F), ktune_id(9F),
ktune_isauto(9F), ktune_isdefault(9F), ktune_name(9F), ktune_pending(9F), ktune_register_handler(9F),
ktune_savedefault(9F), ktune_simple_constraint(9F), ktune_simple_dynamic(9F),
ktune_unregister_handler(9F), ktune_validate_powerof2(9F), ktune_validate_zero_or_min(9F),
ktune_warning(9F).

NOTES

Pre-written tunable handlers are available for common cases. The ktune_validate_powerof2
function validates that a proposed value of a tunable is a power of two. The
ktune_validate_zero_or_min function validates that a proposed value of a tunable is either
zero or greater than some minimum value specified when the handler is registered. The
ktune_simple_constraint function validates that the relationship between two tunable
values meets a simple constraining equation specified when the handler is registered. The
ktune_simple_dynamic function implements dynamic behavior for an integer tunable that
requires no locking. For details on each, see their respective manpages.
NAME
ktune_id - Get the ID of a tunable.

SYNOPSIS
#include <sys/ktune.h>
kttune_id_t ktune_id(
    const char *tunable
);

PARAMETERS
tunable Name of the tunable.

DESCRIPTION
This function returns the tunable ID for the tunable with the given name.

RETURN VALUES
KTUNE_ID_NULL No tunable was found with the specified name.

CONSTRAINTS
Do not call while holding a spinlock with order >= KTUNE_LISTLOCK_ORDER.

EXAMPLES
ktune_id_t tuneid = ktune_id("maxuprc");

SEE ALSO
ktune_get(9F), ktune_name(9F).
**ktune_inactive(9F)**

**NAME**

ktune_inactive - Marks a tunable as no longer in use.

**SYNOPSIS**

```
#include <sys/ktune.h>
void ktune_inactive(
    ktune_id_t tuneid
);
```

**PARAMETERS**

- `tuneid`  
  ID of the tunable to mark inactive.

**DESCRIPTION**

This function tells the tunable infrastructure that the tunable with the specified ID is no longer in use by kernel code. Typically this is only called by a DLKM during the module unload process. The tunable ID can be obtained by calling `ktune_id`.

This function does not remove a tunable from the list of tunables seen by an administrator. However, it is marked as inactive in such lists.

The tunable infrastructure will allow, and apply immediately, any change to an inactive tunable that is consistent with the tunable’s min and max boundaries, if any. No other validation is performed until the tunable is next activated.

All tunables are considered inactive at boot time until they are activated. Tunables are activated by calling `ktune_get`.

No handlers may be registered for a tunable when it is made inactive.

**RETURN VALUES**

None

**CONSTRAINTS**

Do not call while holding a spinlock with order >= `KTUNE_LISTLOCK_ORDER`.

**EXAMPLES**

```
ktune_inactive(tuneid);
```

**SEE ALSO**

`ktune_get(9F), ktune_id(9F)`. 
ktune_isauto(9F)

NAME

ktune_isauto - Checks whether a tunable is being automatically tuned.

SYNOPSIS

#include <sys/ktune.h>

int ktune_isauto(
    ktune_id_t tuneid
);

PARAMETERS

tuneid ID of the tunable to check.

DESCRIPTION

This function checks whether the tunable with the specified ID is being automatically tuned. A tunable is considered to be automatically tuned if it is in its default state (see ktune_isdefault(9F)), has one or more registered handlers capable of computing an algorithmic default value, and has one or more registered handlers capable of making dynamic changes to the tunable's value.

RETURN VALUES

<>0 Tunable is being automatically tuned.
0 Tunable is not being automatically tuned.

CONSTRAINTS

Do not call while holding a spinlock with order => KTUNE_LISTLOCK_ORDER.

EXAMPLES

if (ktune_isauto(tuneid)) {
    ...
}

SEE ALSO

ktune_canauto(9F) ktune_id(9F), ktune_isdefault(9F).
NAME

ktune_isdefault - Checks whether a tunable is in its default state.

SYNOPSIS

#include <sys/ktune.h>

int ktune_isdefault(
    ktune_id_t tuneid
);

PARAMETERS

tuneid  ID of the tunable to check.

DESCRIPTION

This function checks whether the tunable with the specified ID is in its default state. Note that
the default state for the tunable may be a fixed default value, or it may be a self-tuning algorithm;
you cannot tell by using this function. (To find out whether a tunable is self-tuning, use
ktune_isauto.)

RETURN VALUES

<>0  Tunable is in its default state.

0    Tunable is not in its default state.

CONSTRAINTS

Do not call while holding a spinlock with order \( \geq \) KTUNE_LISTLOCK_ORDER.

EXAMPLES

if (ktune_isdefault(tuneid)) {
    ...
}

SEE ALSO

ktune_id(9F), ktune_isauto(9F).
NAME

ktune_isdynamic - Checks whether a tunable can be tuned without a reboot.

SYNOPSIS

#include <sys/ktune.h>

int ktune_isdynamic(
    ktune_id_t tuneid
);

PARAMETERS

tuneid  ID of the tunable to check.

DESCRIPTION

This function checks whether the tunable with the specified ID can be tuned without a reboot. A tunable is considered to be dynamically tuned if it has a registered handler capable of making dynamic changes to the tunable’s value (The handler supports KTOP_COMMIT. See ktune_handler(9F) for information).

RETURN VALUES

<>0  Tunable changes take effect immediately.

0    Tunable changes take effect at next boot.

CONSTRAINTS

Do not call while holding a spinlock with order >= KTUNE_LISTLOCK_ORDER.

EXAMPLES

if (ktune_isdynamic(tuneid)) {
    ...
}

SEE ALSO

ktune_id(9F), ktune_handler(9F).
**NAME**

ktune_name - Get the name(s) of a tunable.

**SYNOPSIS**

```
#include <sys/ktune.h>
const char *ktune_name(
    ktune_id_t tuneid
);
```

**PARAMETERS**

- `tuneid` ID of the tunable whose name is needed.

**DESCRIPTION**

This function returns a pointer to a character string identifying the tunable with the given ID. This string is appropriate to use in error and warning messages. (See `ktune_error(9F)` and `ktune_warning(9F)`.)

This function is used to make the handler functions generic to be used for many tunables at the same time.

The caller must not modify the string.

**RETURN VALUES**

Pointer to the string containing the tunable name.

**CONSTRAINTS**

This function must only be called from a tunable handler.

**EXAMPLES**

```
ktune_error(KTUNE_TXN_NULL,
    "%s has a bad value",
    ktune_name(tuneid));
```

**SEE ALSO**

`ktune_error(9F), ktune_id(9F), ktune_warning(9F)`. 
NAME

ktune_pending - Get the pending value of a tunable.

SYNOPSIS

#include <sys/ktune.h>

int ktune_pending(
    ktune_txn_id_t txnid,
    ktune_id_t tuneid,
    uint64_t *value
);

PARAMETERS

txnid   ID of the transaction in flight, if any.
tuneid  ID of the tunable of interest.
value   Pointer to a variable into which the pending value of the tunable will be placed.

DESCRIPTION

This function retrieves the pending value of a tunable: the value that the tunable will have if the
specified tunable change transaction completes successfully. More specifically:

1. If the transaction involves changing this tunable to a specific value, that value will be
   returned.
2. If the transaction will be applied to take effect at next boot, then the value that will be used
   at next boot will be returned.
3. If the transaction will be applied to the running kernel and does not involve the tunable at
   all, its current value will be returned.
4. If the transaction involves setting this tunable to its default state, the default value will be
   returned.
5. If no transaction is specified (the txnid parameter is KTUNE_TXN_NULL), the current value
   will be returned.

This function is for use within tunable handlers only.

To retrieve the current value of a tunable, regardless of any transaction in flight, call
ktune_current.

RETURN VALUES

0   Success
<>0   Error code returned from tunable handler.

CONSTRAINTS

This function must only be called from a tunable handler.

EXAMPLES

/* From within a tunable handler: */
ret = ktune_pending(event_data->kte_txnid,
             event_data->kte_tuneid,
             &pendingval);
if (ret)
    ...

SEE ALSO

tune_current(9F), ktune_event_t(9F), ktune_get(9F), ktune_handler(9F), ktune_id(9F).
ktune_register_handler(9F)

NAME

ktune_register_handler - Register a handler function for a kernel tunable.

SYNOPSIS

```
#include <sys/ktune.h>

int ktune_register_handler(
    int version,
    const char *tunable,
    int ken_flags,
    int ken_order,
    const char *description,
    ktune_handler_t handler,
    void *handler_data
);
```

PARAMETERS

- **version**: Must be set to KTUNE_VERSION.
- **tunable**: Name of the tunable for which to register the handler.
- **ken_flags**: Must be set to zero.
- **ken_order**: Must be set to KEN_UNORDERED.
- **description**: Description of the handler (for debugging purposes).
- **handler**: Function pointer of the handler.
- **handler_data**: Data to be passed to the handler each time it is called.

DESCRIPTION

This function registers a tunable handler for the specified tunable. For details on tunable handlers, see `ktune_handler(9F)`.

All tunable handlers must be registered before calling `ktune_get` to activate the tunable.

RETURN VALUES

- **0**: success
- **-1**: invalid data
- everything else: a KEN_* error code

CONSTRAINTS

May be called in user context or during boot.

Do not call in interrupt context.

Do not call while holding a spinlock.

EXAMPLES

```c
/* Register a power-of-two validation handler for
 * the tunable vas_hash_locks.
 */
ret = ktune_register_handler(KTUNE_VERSION, "vas_hash_locks",
    0, KEN_UNORDERED,
    "vas_hash_locks validate power of 2",
    (ktune_handler_t)ktune_validate_powerof2,
    NULL);
```
WARNINGS
The handler will be called twice before this function returns. It will be called with a reason code of `KEN_EVENT` and an operation code of `KTOP_CAPABLE`. It will also be called with a reason code of `KEN_REGISTER`.

SEE ALSO
`ktune_get(9F), ktune_handler(9F), ktune_unregister_handler(9F).`
NAME

ktune_savedefault - Save a new default value for a tunable.

SYNOPSIS

#include <sys/ktune.h>

void ktune_savedefault(
    ktune_txn_id_t txnid,
    ktune_id_t tuneid,
    uint64_t defvalue
);

PARAMETERS

txnid       ID of the transaction in progress, if any; otherwise, KTUNE_TXN_NULL.
tuneid      ID of the tunable that has a new default value.
defvalue    new default value for the tunable.

DESCRIPTION

This function saves a new default value for a tunable. This function would be called by a tunable handler that was asked to compute a new default value (a KTOP_GETDEFAULT request). The txnid and tuneid parameters must be those passed to the handler in its ktune_event_t structure.

RETURN VALUES

None

CONSTRAINTS

This function must only be called from a tunable handler.

EXAMPLES

/* in a tunable handler */
case KTOP_GETDEFAULT:
    ktune_savedefault(event_data->kte_txnid,
                       event_data->kte_tuneid,
                       new_default_value);
    break;

SEE ALSO

ktune_event_t(9F), ktune_handler(9F), ktune_id(9F).
**NAME**

ktune_simple_constraint - Handler to validate tunables related by a constraint.

**SYNOPSIS**

```c
#include <sys/ktune.h>

int ktune_simple_constraint(
    ken_id_t eventid,
    int reason,
    ken_instance_t instance,
    ktune_simple_constraint_data_t *constraint,
    ktune_event_t *event_data,
    int *result
);
```

**PARAMETERS**

- `eventid`: The ID of the event.
- `reason`: Reason the handler was invoked:
  - `KEN_REGISTER`: The handler is being registered.
  - `KEN_EVENT`: The handler needs to perform a tunable operation.
  - `KEN_BACKOUT`: The handler needs to reverse a tunable operation.
  - `KEN_UNREGISTER`: The handler is being unregistered.
- `instance`: ID of this particular invocation of this handler.
- `constraint`: Pointer to the `ktune_simple_constraint_data_t` structure which defines the constraint between the 2 tunables.
- `event_data`: When the handler is invoked with `reason` set to `KEN_EVENT` or `KEN_BACKOUT`, this parameter points to a `ktune_event_t` structure describing the tunable operation to be performed or reversed. For other reason values, this parameter is `NULL`.
- `result`: Points to an integer variable into which the handler must store its result code.

**DESCRIPTION**

This is a generic tunable handler that validates any constraint between two tunables that can be written in the form: \((\text{scale}_a \times \text{tunable}_a) \leq (\text{scale}_b \times \text{tunable}_b) + \text{offset}\).

This handler requires that the tunables are of integer type. This tunable handler supports the KTOP_VALIDATE operation.

While registering the handler, the address of the `ktune_simple_constraint_data_t` structure describing the constraint must be supplied as the `handler_data` parameter to `ktune_register_handler`. If the result parameter points to the value zero after the handler is called, the values for the tunables satisfy the constraint. Note that this handler must not be explicitly registered if the constraint is specified in the `modmeta` file for the module defining the tunables.

**RETURN VALUES**

- `KEN_DONE`: Handler functions always return `KEN_DONE`.

**CONSTRAINTS**

See `ktune_handler(9F)` for information on constraints for tunable handlers.
EXAMPLES

To validate the constraint `tunable_a < tunable_b`:

```c
#include <sys/ktune.h>
static ktune_simple_constraint_data_t constraint;
{
    int ret
    ...
    constraint.ksc_name_a = "tunable_a";
    constraint.ksc_name_b = "tunable_b";
    constraint.ksc_scale_a = 1;
    constraint.ksc_scale_b = 1;
    constraint.ksc_offset  = -1;
    ret = ktune_register_handler(KTUNE_VERSION, "tunable_a",
                          0, KEN_UNORDERED,
                          "Constraint handler for tunable_a",
                          (ktune_handler_t)ktune_simple_constraint,
                          &constraint);

    ret = ktune_register_handler(KTUNE_VERSION, "tunable_b",
                          0, KEN_UNORDERED,
                          "Constraint handler for tunable_b",
                          (ktune_handler_t)ktune_simple_constraint,
                          &constraint);
    ...
}
```

To validate the constraint `2*tunable_a <= tunable_b`:

```c
#include <sys/ktune.h>
static ktune_simple_constraint_data_t constraint;
{
    int ret
    ...
    constraint.ksc_name_a = "tunable_a";
    constraint.ksc_name_b = "tunable_b";
    constraint.ksc_scale_a = 2;
    constraint.ksc_scale_b = 1;
    constraint.ksc_offset  = 0;
    ret = ktune_register_handler(KTUNE_VERSION, "tunable_a",
                          0, KEN_UNORDERED,
                          "Constraint handler for tunable_a",
                          (ktune_handler_t)ktune_simple_constraint,
                          &constraint);

    ret = ktune_register_handler(KTUNE_VERSION, "tunable_b",
                          0, KEN_UNORDERED,
                          "Constraint handler for tunable_b",
                          (ktune_handler_t)ktune_simple_constraint,
                          &constraint);
```
}  

**SEE ALSO**

ktune_handler(9F), ktune_event_t(9F), ktune_simple_constraint_data_t(9F), ktune_register_handler(9F).
NAME

ktune_simple_constraint_data_t - Details for the simple constraint handler.

SYNOPSIS

#include <sys/ktune.h>

typedef struct ktune_simple_constraint_data {
    const char *ksc_name_a;
    const char *ksc_name_b;
    int64_t      ksc_scale_a;
    int64_t      ksc_scale_b;
    int64_t      ksc_offset;
    ktune_id_t   ksc_tuneid_a;
    ktune_id_t   ksc_tuneid_b;
} ktune_simple_constraint_data_t;

DESCRIPTION

The ktune_simple_constraint_data_t structure is used to represent the constraint expression that relates two tunables. A constraint expression can take the following form:

(scale_a * tunable_a) <= (scale_b * tunable_b) + offset

The ktune_simple_constraint_handler tunable handler uses a pointer to the ktune_simple_constraint_data_t structure to validate the constraint expression. Either the tunable name or tunable ID fields in the structure need to be filled in. If the tunable ID fields are not filled in, they must be set to KTUNE_ID_NULL.

STRUCTURE MEMBERS

ksc_name_a   The name of the first tunable in the constraint expression.
ksc_name_b   The name of the second tunable in the constraint expression.
ksc_scale_a  The scaling factor for the first tunable.
ksc_scale_b  The scaling factor for the second tunable.
ksc_offset   The offset to be applied.
ksc_tuneid_a The tunable ID for the first tunable.
ksc_tuneid_b The tunable ID for the second tunable.

SEE ALSO

ktune_handler(9F), ktune_simple_constraint(9F)
ktune_simple_dynamic(9F)

NAME

ktune_simple_dynamic - Allows dynamic changes to an integer tunable that needs no locking.

SYNOPSIS

#include <sys/ktune.h>

int ktune_simple_dynamic(
    ken_id_t eventid,
    int reason,
    ken_instance_t instance,
    int *tune_var,
    ktune_event_t *event_data,
    int *result
);

PARAMETERS

eventid    The ID of the event.
reason     Reason the handler was invoked:
            KEN_REGISTER   The handler is being registered.
            KEN_EVENT     The handler needs to perform a tunable operation.
            KEN_BACKOUT   The handler needs to reverse a tunable operation.
            KEN_UNREGISTER The handler is being unregistered.
instance   ID of this particular invocation of this handler.
tune_var   Pointer to the tunable variable.
event_data When the handler is invoked with reason set to KEN_EVENT or KEN_BACKOUT,
            this parameter points to a ktune_event_t structure describing the tunable
            operation to be performed or reversed. For other reason values, this parameter
            is NULL.
result     Points to an integer variable into which the handler must store its result code.

DESCRIPTION

This is a generic tunable handler to make a tunable dynamic. This handler does not use any
locking while changing the value of the tunable. This handler must be registered only for tunables
that are of integer type. This tunable handler supports the KTOP_COMMIT operation.

While registering the handler, the address of integer variable with the tunable value must be
supplied as the handler_data parameter to ktune_register_handler.

RETURN VALUES

KEN_DONE Handler functions always return KEN_DONE.

CONSTRAINTS

See ktune_handler(9F) for information on constraints for tunable handlers.

EXAMPLES

#include <sys/ktune.h>
int mytunable;

{                      
    int ret

... ret = ktune_register_handler(KTUNE_VERSION, "mytunable",
0, KEN_UNORDERED,
"Dynamic handler for mytunable",

(ktune_handler_t)ktune_simple_dynamic,

&mytunable);
...

SEE ALSO

ktune_current(9F), ktune_handler(9F), ktune_event_t(9F), ktune_register_handler(9F).
ktune_unregister_handler(9F)

NAME

ktune_unregister_handler - Unregister a handler function for a kernel tunable.

SYNOPSIS

#include <sys/ktune.h>

int ktune_unregister_handler(
    int version,
    const char *tunable,
    int ken_flags,
    int ken_order,
    ktune_handler_t handler,
    void *handler_data
);

PARAMETERS

version  Must be set to KTUNE_VERSION.
tunable  Name of the tunable from which to unregister the handler.
ken_flags Must be set to the same value given in the ktune_register_handler call when the handler was registered.
ken_order Must be set to the same value given in the ktune_register_handler call when the handler was registered.
handler  Function pointer of the handler to unregister.
handler_data Must be set to the same value given in the ktune_register_handler call when the handler was registered.

DESCRIPTION

This function unregisters a tunable handler for the specified tunable. The tunable handlers registered by a module need to be unregistered by the module unload function.

RETURN VALUES

0            success
-1           invalid data
everything else a KEN_* error code

CONSTRAINTS

May be called in user context or during boot.
Do not call in interrupt context.
Do not call while holding a spinlock.

EXAMPLES

/* Unregister a handler for the tunable vas_hash_locks. */
ret = ktune_unregister_handler(KTUNE_VERSION, "vas_hash_locks",
    0, KEN_UNORDERED,
    (ktune_handler_t)ktune_validate_powerof2,
    NULL);
WARNINGS

The handler will be called twice before this function returns. It will be called with a reason code of KEN_EVENT and an operation code of KTOP_CAPABLE. It will also be called with a reason code of KEN_UNREGISTER.

SEE ALSO

ktune_handler(9F), ktune_inactive(9F), ktune_register_handler(9F).
NAME

ktune_validate_powerof2 - Handler to validate that the tunable is a power of 2.

SYNOPSIS

#include <sys/ktune.h>

int ktune_validate_powerof2(
    ken_id_t eventid,
    int reason,
    ken_instance_t instance,
    void *unused,
    ktune_event_t *event_data,
    int *result
);

PARAMETERS

eventid The ID of the event.
reason Reason the handler was invoked:
KEN_REGISTER The handler is being registered.
KEN_EVENT The handler needs to perform a tunable operation.
KEN_BACKOUT The handler needs to reverse a tunable operation.
KEN_UNREGISTER The handler is being unregistered.
instance ID of this particular invocation of this handler.
unused The handler data is not used in this handler.
event_data When the handler is invoked with reason set to KEN_EVENT or KEN_BACKOUT, this parameter points to a ktune_event_t structure describing the tunable operation to be performed or reversed. For other reason values, this parameter is NULL.
result Points to an integer variable into which the handler must store its result code.

DESCRIPTION

This is a generic tunable handler which validates that the tunable is a power of 2. This tunable handler requires the tunable to be of integer type. This tunable handler supports the KTOP_VALIDATE operation. If the result parameter points to the value zero after the handler is called, the tunable is a power of 2. Note that this handler must not be explicitly registered if the powerof2 constraint is specified in the tunable definition in the modmeta file.

RETURN VALUES

KEN_DONE Handler functions always return KEN_DONE.

CONSTRAINTS

See ktune_handler(9F) for information on constraints for tunable handlers.

EXAMPLES

#include <sys/ktune.h>
{
    int ret;
    ...
    ret = ktune_register_handler(KTUNE_VERSION, "mytunable",
0, KEN_UNORDERED,
"power of 2 handler for mytunable",
(ktune_handler_t)ktune_validate_powerof2,
NULL);

...

SEE ALSO

ktune_handler(9F), ktune_event_t(9F), ktune_register_handler(9F).
NAME

ktune_validate_zero_or_min - Handler to allow zero as a valid value for the tunable.

SYNOPSIS

#include <sys/ktune.h>

int ktune_validate_zero_or_min(
    ken_id_t eventid,
    int reason,
    ken_instance_t instance,
    uint64_t *minimum,
    ktune_event_t *event_data,
    int *result
);

PARAMETERS

- **eventid**: The ID of the event.
- **reason**: Reason the handler was invoked, one of:
  - KEN_REGISTER: The handler is being registered.
  - KEN_EVENT: The handler needs to perform a tunable operation.
  - KEN_BACKOUT: The handler needs to reverse a tunable operation.
  - KEN_UNREGISTER: The handler is being unregistered.
- **instance**: ID of this particular invocation of this handler.
- **minimum**: The alternate minimum value for the tunable.
- **event_data**: When the handler is invoked with **reason** set to KEN_EVENT or KEN_BACKOUT, this parameter points to a ktune_event_t structure describing the tunable operation to be performed or reversed. For other reason values, this parameter is NULL.
- **result**: Points to an integer variable into which the handler must store its result code.

DESCRIPTION

This is a generic tunable handler that allows the tunable to have the value zero even if it falls outside the valid range for the tunable. This handler requires the tunable to be of integer type. This tunable handler supports the KTOP_VALIDATE operation. If the result parameter points to the value zero after the handler is called, tunable value is valid. Note that this handler must not be explicitly registered if the zero_ok constraint is specified in the tunable definition in the modmeta file.

RETURN VALUES

KEN_DONE: Handler functions always return KEN_DONE.

CONSTRAINTS

See ktune_handler(9F) for information on constraints for tunable handlers.

EXAMPLES

#include <sys/ktune.h>
{
    int ret;
    uint64_t minimum=10;
ret = ktune_register_handler(KTUNE_VERSION, "mytunable",
          0, KEN_UNORDERED,
          "zero_ok handler for mytunable",
          (ktune_handler_t)ktune_validate_zero_or_min,
          &(minimum));

SEE ALSO

ktune_handler(9F), ktune_event_t(9F), ktune_register_handler(9F).
ktune_warning(9F)

NAME

ktune_warning - Log a tunable error message.

SYNOPSIS

#include <sys/ktune.h>

void ktune_warning(
    ktune_txn_id_t txnid,
    const char *format,
    ...
);

PARAMETERS

txnid    ID of the transaction in flight, if any.
format    printf-style format string.
...    Arguments needed for format string.

DESCRIPTION

This function logs a tunable change warning message. If a tunable change transaction is in
progress, the warning message will be returned to the caller. Otherwise, it will be logged to the
system message buffer. This function must only be called from a tunable handler.

Do not embed tunable names in warning message strings. Instead, put a %s where the name is
to go, and use ktune_name as an argument to fill in the name.

RETURN VALUES

None

CONSTRAINTS

This function must only be called from a tunable handler.

EXAMPLES

/* From within a tunable handler: */
ktune_warning(event_data->kte_txnid,
    "%lu is probably not a desirable value for %s.",
    value, ktune_name(event_data->kte_tuneid));

SEE ALSO

ktune_error(9F), ktune_event_t(9F), ktune_handler(9F), ktune_id(9F), ktune_name(9F).
NAME

ldsid - Return (load) the space ID (SID) for a kernel virtual address.

SYNOPSIS

#include <sys/kern_svcs.h>
space_t ldsid(
    void *addr
);

PARAMETERS

addr    Kernel virtual address

DESCRIPTION

The ldsid kernel function returns the space ID (SID) for the kernel virtual address (addr). The SID and virtual address are combined to form the global virtual address.

Use ldsid with kernel virtual addresses only. Do not call ldsid for a user space address.

RETURN VALUES

The ldsid function returns the returns the space ID (SID) for a kernel virtual address.

CONSTRAINTS

None

EXAMPLE

/**
 * Copy data into the buffer specified through a buf (bp).
 * The b_spaddr and b_un.b_addr fields may contain the SID
 * and virtual address of a user space buffer. The from_addr
 * is a kernel virtual address and ldsid returns its SID.
 */
privlbcopy(ldsid(from_addr), from_addr,
    bp->b_spaddr, bp->b_un.b_addr,
    sizeof(*from_addr));

SEE ALSO

privlbcopy(9F)
major(9F)

NAME
major - Extract the major number from a device number.

SYNOPSIS
#include <sys/sysmacros.h>

#define major(x) ((int)(((unsigned)(x)>>24)&0xff))

PARAMETERS
x A dev_t device number.

DESCRIPTION
The major kernel macro returns the major number from the device number.

CONSTRAINTS
None

SEE ALSO
minor(9F)
NAME
makedev - Make a device number from major and minor numbers.

SYNOPSIS
#include <sys/sysmacros.h>

#define makedev(x, y) ((dev_t)(((x)<<24) | (y & 0xffffffff))

PARAMETERS
x  A major number.
y  A minor number.

DESCRIPTION
The makedev kernel macro builds a device number from major and minor numbers.

CONSTRAINTS
None

SEE ALSO
major(9F), minor(9F)
MALLOC(9F)

NAME
MALLOC – Kernel macro to allocate kernel memory.

SYNOPSIS
#include <sys/malloc.h>
#define MALLOC(space, cast, size, type, flags)

PARAMETERS
space   Pointer to the kernel memory allocated.
cast    Type of the space pointer to be used as a cast in the macro.
size    The number of bytes (size) of kernel memory to allocate.
type    The memory allocation pool type.
flags   Flag to indicate the caller cannot block and wait for kernel memory availability.

DESCRIPTION
The MALLOC and FREE kernel macros are deprecated interfaces and may be obsoleted in a future release of HP-UX. Use the kernel functions kmalloc and kfree in place of MALLOC and FREE.
The MALLOC kernel macro allocates size bytes of kernel memory from the pool type specified. The memory allocated will always be contiguous in the virtual address space, but may be discontiguous in the physical address space. If size is greater than PAGESIZE, the physical pages allocated will probably not be physically contiguous.
Drivers must allocate memory from pool type M_IOSYS or M_DMA. The M_IOSYS pool is for general I/O purposes; the M_DMA pool is for DMA purposes.
The flags parameter may optionally have the M_NOWAIT flag set. If M_NOWAIT is set and no memory is available from the requested pool type, the functions will return NULL. Without M_NOWAIT set, the caller can be blocked and made to wait for memory to become available.
You must set the M_NOWAIT flag if MALLOC is:
- Called in an interrupt context, or
- Called while holding a spinlock
If M_NOWAIT is set, the caller must be prepared to handle the case where no kernel memory has been allocated.
M_WAITOK must be passed as the flags parameter if the caller is able to block and wait for memory to become available.

RETURN VALUES
The MALLOC returns the following values to the space pointer:
<>NULL   Virtual address of kernel memory allocated.
NULL     No kernel memory has been allocated. NULL is returned only when the M_NOWAIT flag is set; otherwise, the caller is blocked and waits until memory becomes available.

CONSTRAINTS
If the M_NOWAIT flag is not set:
- The caller must not be in an interrupt context.
- No spinlocks can be held.
EXAMPLE

my_struct_t * my_addr;

MALLOC (my_addr, my_struct_t *, sizeof(my_struct_t),
       M_IOSYS, M_NOWAIT);

if (!my_addr) {
    /*
     * Error!  No kernel memory currently available.
     */
}

SEE ALSO

FREE(9F), kfree(9F), kmalloc(9F)
map_mem_to_host(9F)

NAME

map_mem_to_host - Map physical bus address to host virtual space.

SYNOPSIS

#include <sys/wsio.h>
caddr_t map_mem_to_host(
    struct isc_table_type *isc,
    caddr_t phys_addr,
    size_t size
);

PARAMETERS

isc ISC pointer that corresponds to the interface card associated with this memory.
phys_addr Base physical address for a range of memory space on an I/O bus.
size Size of the memory range in bytes.

DESCRIPTION

The kernel function maps physical bit address to a host virtual address for cards with memory ranges. Hardware disables caching for all EISA data accesses, including EISA memory space. The map_mem_to_host function returns NULL if a request overlaps an existing mapping.

RETURN VALUES

<>NULL Successful completion. The value is the host virtual address for accessing the space specified by the parameters.

NULL Error.

CONSTRAINTS

None

SEE ALSO

unmap_mem_from_host(9F), kernel_iomap(9F), kernel_iounmap(9F)
minor(9F)

NAME
minor - Extract the minor number from a device number.

SYNOPSIS
#include <sys/sysmacros.h>

#define minor(x) ((long)((x)&0xffffffff))

PARAMETERS
x A dev_t device number.

DESCRIPTION
The minor kernel macro returns the minor number from a device number.

CONSTRAINTS
None

SEE ALSO
major(9F)

194 Kernel Reference Pages
NAME
minphys - Limit the b_bcount field in a buf structure to the value MAXPHYS.

SYNOPSIS
#include <sys/buf.h>
void minphys(
    struct buf *bp
);

PARAMETERS
bp Pointer to a buf structure.

DESCRIPTION
The minphys kernel function compares bp->b_bcount against the MAXPHYS value defined in <sys/param.h>. If bp->b_bcount is greater than MAXPHYS, bp->b_bcount is changed to MAXPHYS.

The minphys is passed as the mincnt parameter to physio. In this way, physio can break a large data transfer into multiple smaller transfers, each no greater than MAXPHYS bytes in length.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
int mydriver_read(dev_t dev, struct uio * uio)
{
    return physio(mydriver_strategy, NULL, dev, B_READ,
                   minphys, uio);
}

SEE ALSO
physio(9F), driver_minphys(9E)
msg_printf(9F)

NAME

msg_printf - Write diagnostic information to the kernel message buffer.

SYNOPSIS

#include <sys/kern_svcs.h>

int msg_printf(
    const char *format,
    ...);

PARAMETERS

format A set of printing characters and limited conversion specifications, as defined in printf.

DESCRIPTION

The msg_printf kernel function is a scaled down version of the C library printf routine (see printf(3S)).

The msg_printf function writes diagnostic information to the msgbuf kernel message buffer only and not to the console.

The msg_printf kernel routine can accept the following formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Printed Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>%%</td>
<td>%; no argument</td>
</tr>
<tr>
<td>%b</td>
<td>Characters from string argument; can include backslash-escape codes</td>
</tr>
<tr>
<td>%c</td>
<td>Character from integer argument</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal from integer argument</td>
</tr>
<tr>
<td>%lx</td>
<td>The argument x is long integer; x is one of x, d, u, or o</td>
</tr>
<tr>
<td>%o</td>
<td>Octal from integer argument</td>
</tr>
<tr>
<td>%s</td>
<td>Characters from string argument</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal from integer argument</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal from integer argument</td>
</tr>
</tbody>
</table>

These formats are the same as in printf(3S). Other formats specified in printf(3S) are not supported.

RETURN VALUES

The msg_printf function returns the length of the formatted string.

CONSTRAINTS

None

SEE ALSO

printf(9F), printf(3S), vsprintf(9F), vsnprintf(9F), sprintf(3S)
ms_gettimeofday(9F)

NAME

ms_gettimeofday - Get the current date and time with microsecond accuracy

SYNOPSIS

#include <sys-clock.h>
#include <sys/_timeval.h>
struct timeval ms_gettimeofday(void);

DESCRIPTION

The ms_gettimeofday function returns the current time, expressed as seconds and microseconds since 00:00 Coordinated Universal Time (UTC), January 1, 1970.

The time returned is not guaranteed to only increase because it is possible to set the system date and time to any arbitrary value from a user space program using the stime(2) system call.

RETURN VALUES

On successful return, timeval contains the current number of seconds and microseconds that have elapsed since 00:00 UTC. January 1, 1970. If this function is called before the machdep clock subsystem initialization, timeval is 0.

CONSTRAINTS

Can be called in interrupt context.
Can be called while holding a spinlock.

EXAMPLES

#include <sys-clock.h>   /* for ms_gettimeofday prototype */
#include <sys/_timeval.h> /* for struct timeval definition */

/**
 * Obtain the current time
 */

struct timeval current_time = ms_gettimeofday();

SEE ALSO

get_system_time(9F)
mutex_alloc(9F)

NAME

mutex_alloc - Dynamically allocate and initialize a mutex

SYNOPSIS

#include <sys/mutex.h>

mutex_t mutex_alloc(
    char *name,
    mutex_attr_t *attr,
    mutex_attr_flg_t alloc_flag,
    unsigned int major_order,
    uintptr_t minor_order
);

PARAMETERS

name The mutex name.
attr Pointer to a mutex attribute (mutex_attr_t) structure or NULL.
alloc_flag MTX_WAITOK or MTX_NOWAIT; these are mutually exclusive.
major_order The major order number that is used for deadlock detection in a debug kernel.
minor_order The order for locks with the same major order number. Frequently, the address of the structure being protected.

DESCRIPTION

The mutex_alloc kernel function dynamically allocates and initializes a mutex.

Mutexes support the following attributes:

- MTX_ATTR_PRIOCEIL: The mutex participates in the priority ceiling protocol.
- MTX_ATTR_NOSWAP: The thread holding the lock will not be deactivated.
- MTX_ATTR_DISOWNABLE: The mutex can be disowned.
- MTX_ATTRSAME_ORDER_OK: This corresponds to the deadlock safe feature of beta semaphores. The thread can hold other locks of the same major and minor order.

In the future, additional flags might be supported that provide performance hints to the mutex routines.

RETURN VALUES

<>NULL Pointer to the allocated mutex.
NULL The allocation failed because of a lack of memory (in the case of MTX_NOWAIT).

CONSTRAINTS

Must be called in a thread context and with no spinlocks held, if MTX_WAITOK is specified.

If attr is a pointer, the attribute structure must be initialized by mutex_attr_init and mutex_attr_setflag before calling this routine.

EXAMPLE

#include <sys/mutex.h>

mutex_t *count_lock_p;
/*
* Use the mutex.
*/
count_lock_p = mutex_alloc("My Mutex Lock", &my_attr, MTX_WAITOK,
                         MY_MAJOR_NUMBER, NULL);
/*
 * Done with the mutex.
 */
mutex_dealloc(count_lock_p);

SEE ALSO

mutex_attr_init(9F), mutex_attr_setflag(9F), mutex_dealloc(9F), mutex_lock(9F), mutex_owned(9F),
mutex_trylock(9F), mutex_unlock(9F)
mutex_dealloc(9F)

NAME

mutex_dealloc - Destroy and deallocate a mutex

SYNOPSIS

#include <sys/mutex.h>

void mutex_dealloc(
    mutex_t *mp
);

PARAMETERS

mp  Pointer to the mutex to deallocate.

DESCRIPTION

The mutex_dealloc kernel function destroys and deallocates a mutex.

RETURN VALUES

None

CONSTRAINTS

Must be called with the mutex unlocked.

EXAMPLE

#include <sys/mutex.h>

mutex_t *count_lock_p;

/*
 * Use the mutex.
 */
count_lock_p = mutex_alloc ("My Mutex Lock", &my_attr, MTX_WAITOK,
    MY_MAJOR_NUMBER, NULL);

/*
 * Done with the mutex.
 */
mutex_dealloc (count_lock_p);

SEE ALSO

mutex_alloc(9F), mutex_attr_init(9F), mutex_attr_setflag(9F), mutex_lock(9F), mutex_owned(9F),
mutex_trylock(9F), mutex_unlock(9F)
NAME
mutex_attr_init - Initializes the mutex attribute structure to its default value.

SYNOPSIS
#include <sys/mutex.h>

void mutex_attr_init(
    mutex_attr_t *attr,
    size_t size
);

PARAMETERS
attr Pointer to a mutex attribute (mutex_attr_t) structure.
size The size of the mutex attribute structure.

DESCRIPTION
The mutex_attr_init kernel function initializes the mutex attribute structure to its default value. Kernel modules can pass optional attributes to the mutex creation routines through an attribute data structure. Before using this structure, kernel modules must initialize the structure to its default values. After the data structure is initialized, the module can set the specific attributes for the mutex with mutex_attr_setflag and then call the creation routine.

This routine is provided for binary compatibility. If new attributes are added to the attribute structure in the future, you will not have to recompile your code.

Mutexes support the following attributes:

MTX_ATTR_PRIOCEIL The mutex participates in the priority ceiling protocol.
MTX_ATTR_NOSWAP The thread holding the lock will not be deactivated.
MTX_ATTR_DISOWNABLE The mutex can be disowned.
MTX_ATTR_SAME_ORDER_OK This corresponds to the deadlock safe feature of beta semaphores. The thread can hold other locks of the same major and minor order.

The default value for the attribute structure is none of the above. After the attribute structure is initialized to its default value, use the mutex_attr_setflag to set specific flags in the attribute structure.

In the future, additional flags might be supported that provide performance hints to the mutex routines.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
#include <sys/mutex.h>

mutex_t *count_lock_p;
mutex_attr_t my_attr;
/*
 * Set default values
 */
mutex_attr_init (&my_attr, sizeof (mutex_attr_t));
/*
 * Turn on priority ceiling and same order okay.
 */
mutex_attr_setflag (&my_attr, MTX_ATTR_PRIOCEIL | MTX_ATTRSAME_ORDER_OK, 
                        MTX_ATTR_FLAG_ON);

count_lock_p = mutex_alloc("My Mutex Lock", &my_attr, MTX_WAITOK,
                             MY_MAJOR_NUMBER, NULL);

SEE ALSO

mutex_alloc(9F), mutex_attr_setflag(9F), mutex_dealloc(9F), mutex_lock(9F), mutex_owned(9F),
mutex_trylock(9F), mutex_unlock(9F)
NAME

mutex_attr_setflag - Set mutex attribute structure values.

SYNOPSIS

#include <sys/mutex.h>

void mutex_attr_setflag(
    mutex_attr_t *attr,
    mutex_attr_flg_t flags,
    mutex_attr_flg_set_t mode
);

PARAMETERS

attr        Pointer to a mutex attribute (mutex_attr_t) structure.
flags       The attributes to turn on or off. This is logical OR of the attributes in the DESCRIPTION section.
mode        The attribute's mode: MTX_ATTR_FLAG_ON (on) or MTX_ATTR_FLAG_OFF (off).

DESCRIPTION

The mutex_attr_setflag kernel function sets the specified attributes on or off in the mutex attribute structure. Kernel modules can pass optional attributes to the mutex creation routines through an attribute data structure. Before using this structure, kernel modules must initialize the structure to its default values. After the data structure is initialized, the module uses mutex_attr_setflag to set the specific attributes for the mutex. Then, the attribute structure is passed to the creation routine.

Mutexes support the following attributes:

- MTX_ATTR_PRIOCEIL: The mutex participates in the priority ceiling protocol.
- MTX_ATTR_NOSWAP: The thread holding the lock will not be deactivated.
- MTX_ATTR_DISOWNABLE: The mutex can be disowned.
- MTX_ATTRSAME_ORDER_OK: This corresponds to the deadlock safe feature of beta semaphores. The thread can hold other locks of the same major and minor order.

By default, all attributes in the attribute structure are off.

In the future, additional flags might be supported that provide performance hints to the mutex routines.

RETURN VALUES

None

CONSTRAINTS

The routine must call mutex_attr_init to initialize the attribute structure before calling mutex_attr_setflag.

EXAMPLE

#include <sys/mutex.h>

mutex_t *count_lock_p;
mutex_attr_t my_attr;
/*
 * Set default values
 */
mutex_attr_init (&my_attr, sizeof (mutex_attr_t));

* Turn on priority ceiling and same order okay.

mutex_attr_setflag (&my_attr, MTX_ATTR_PRIOCEIL | MTX_ATTR_SAME_ORDER_OK, MTX_ATTR_FLAG_ON);

count_lock_p = mutex_alloc ("My Mutex Lock", &my_attr, MTX_WAITOK, MY_MAJOR_NUMBER, NULL);

SEE ALSO

mutex_alloc(9F), mutex_attr_init(9F), mutex_dealloc(9F), mutex_lock(9F), mutex_owned(9F),
mutex_trylock(9F), mutex_unlock(9F)
mutex_lock(9F)

NAME

mutex_lock -- Lock (acquire) a mutex

SYNOPSIS

#include <sys/mutex.h>

void mutex_lock(
    mutex_t *mp
);

PARAMETERS

mp  Pointer to the mutex to lock.

DESCRIPTION

The mutex_lock kernel function acquires the specified mutex. The function will block or spin until the mutex is available. If the thread must wait for the mutex, it will not be awakened by a signal.

RETURN VALUES

None

CONSTRAINTS

Must be called only from a thread context. The thread can neither hold any spinlocks nor already own the specified mutex.

EXAMPLE

#include <sys/mutex.h>

mutex_t *lock_p;
header_t header;

int add_to_list(int flags, data_t *data)
{
    /*
     * Acquire the mutex for my driver.
     */
    mutex_lock(lock_p);
    /* Add to head of single linked list.
     * The list is protected by lock_p.
     */
    data->next = header.head_of_list;
    header.head_of_list = data;
    /*
     * Release the mutex when done.
     */
    mutex_unlock(lock_p);
    return (OK);
}

SEE ALSO

mutex_alloc(9F), mutex_attr_init(9F), mutex_attr_setflag(9F), mutex_dealloc(9F), mutex_owned(9F), mutex_trylock(9F), mutex_unlock(9F)
NAME
mutex_owned - Test whether the thread has the mutex locked

SYNOPSIS
#include <sys/mutex.h>
int mutex_owned(
    mutex_t *mp
);

PARAMETERS
mp A pointer to the mutex to test.

DESCRIPTION
The mutex_owned kernel function tests whether the current thread has the mutex locked.

RETURN VALUES
<>0 The thread owns the mutex.
0 The thread does not own the mutex.

CONSTRAINTS
Must be called from a thread context.

EXAMPLE
#include <sys/mutex_impl.h>
mutex_t *lock_p;

int add_to_list(int flags, data_t *data)
{
    /* Better own the mutex because
     * we are going to update the list.
     */
    VASSERT (mutex_owned(lock_p));

    /* Add to head of single linked list.
     * The list is protected by lock_p.
     */
    data->next = header.head_of_list;
    header.head_of_list = data;

    return (OK);
}

SEE ALSO
mutex_alloc(9F), mutex_attr_init(9F), mutex_attr_setflag(9F), mutex_dealloc(9F), mutex_lock(9F),
mutex_trylock(9F), mutex_unlock(9F)
mutex_trylock(9F)

NAME

mutex_trylock - Conditionally acquire a mutex

SYNOPSIS

#include <sys/mutex.h>

int mutex_trylock(
    mutex_t *mp
);

PARAMETERS

mp  A pointer to the mutex to lock.

DESCRIPTION

The mutex_trylock kernel function conditionally acquires a mutex. If the mutex is not available, the routine fails.

RETURN VALUES

<>0  The mutex lock succeeded.

0    The mutex lock failed..

CONSTRAINTS

None

EXAMPLE

#include <sys/mutex.h>

mutex_t *lock_p;
header_t header;

int add_to_list(int flags, data_t *data)
{
    if (flags == NOWAIT) {
        /*
         * Cannot wait for the lock.
         */
        if !mutex_trylock(lock_p) {
            return (FAILED_TO_DO_WORK);
        }
    }
    else mutex_lock(lock_p);
    /* Add to head of single linked list.
    * The list is protected by lock_p.
    */
    data->next = header.head_of_list;
    header.head_of_list = data;
    /*
    * Release the mutex when done.
    */
    mutex_unlock(lock_p);
    return (OK);
}

SEE ALSO

mutex_alloc(9F), mutex_attr_init(9F), mutex_attr_setflag(9F), mutex_dealloc(9F), mutex_lock(9F),
mutex_owned(9F), mutex_unlock(9F)
**mutex_unlock(9F)**

**NAME**

mutex_unlock - Unlock a mutex

**SYNOPSIS**

```c
#include <sys/mutex.h>
void mutex_unlock(
    mutex_t *mp
);
```

**PARAMETERS**

- `mp` Pointer to the mutex to unlock.

**DESCRIPTION**

The `mutex_unlock` kernel function unlocks the specified mutex. If there are threads waiting for the mutex, it unblocks a waiting thread.

Only the thread that owns the mutex can unlock it. However, if the lock has been disowned, another thread or interrupt handler can unlock it.

**RETURN VALUES**

None

**CONSTRAINTS**

None

**EXAMPLE**

```c
#include <sys/mutex.h>

mutex_t *lock_p;
header_t header;

int add_to_list(int flags, data_t *data)
{
    /* Acquire the mutex for my driver. */
    mutex_lock(lock_p);
    /* Add to head of single linked list. 
    * The list is protected by lock_p. 
    */
    data->next = header.head_of_list;
    header.head_of_list = data;
    /* 
    * Release the mutex when done. 
    */
    mutex_unlock(lock_p);
    return (OK);
}
```

**SEE ALSO**

mutex_alloc(9F), mutex_attr_init(9F), mutex_attr_setflag(9F), mutex_dealloc(9F), mutex_lock(9F), mutex_owned(9F), mutex_trylock(9F)
NAME

p_pgrp - Return the process group identifier for a process.

SYNOPSIS

#include <sys/proc_iface.h>

pid_t p_pgrp(
    proc_t *procp
);

PARAMETERS

procp Pointer to a proc_t structure.

DESCRIPTION

The p_pgrp kernel function returns the process group identifier for a process.

The procp parameter is a pointer to a proc_t structure. Use the procp_self routine to obtain the pointer to the proc structure for the current process.

RETURN VALUES

The p_pgrp kernel function returns the process group identifier for a process.

CONSTRAINTS

None

EXAMPLE

#include <sys/proc.h>
#include <sys/signal.h>
#include <sys/proc_iface.h>

/*
 * Save the pointer to the proc structure
 */
proc_t *my_procp = procp_self();
/*
 * Signal all processes in the current process group.
 */
gsignal(p_pgrp(my_procp),SIGIO);

SEE ALSO

gsignal(9F)
panic(9F)

NAME
panic - Soft-crash the operating system.

SYNOPSIS
#include <sys/kern_svcs.h>
void panic(
    char * str
);

PARAMETERS
str Pointer to a character string message. No format capability.

DESCRIPTION
The panic kernel function prints str to the system console and halts the system.
The panic function prints the processor status register, the program counter register, the trap
type on processor exceptions, and part of the kernel stack.

RETURN VALUES
The panic function does not return.

CONSTRAINTS
None

WARNING
The panic function halts the system and may cause file system damage. Use this call only to
flag catastrophic and unrecoverable failures.
**NAME**

physio - Perform unbuffered, physical I/O data transfers.

**SYNOPSIS**

```c
#include <sys/buf.h>

int physio(
    int (*strat)(),
    struct buf *bp,
    int dev,
    int flag,
    void (*mincnt)(),
    struct uio *uiop
);
```

**PARAMETERS**

- **strat**: Pointer to the driver strategy function.
- **bp**: Pointer to a buf structure.
- **dev**: Device number.
- **flag**: Read/write flag: B_READ or B_WRITE.
- **mincnt**: Function that limits maximum transfer length.
- **uiop**: Pointer to uio structure.

**DESCRIPTION**

The `physio` kernel function performs unbuffered, physical I/O data transfers. It accepts a user I/O request specified by a `uio` structure, prepares the data pages for I/O, builds the associated `buf` structure, calls the specified driver `strat` function, and waits for the I/O to complete.

The `strat` parameter is a pointer to the driver strategy function. The `physio` function calls `strat` to start the I/O transfer, then waits by calling `biowait`. When the I/O transfer completes, `strat` calls `biodone` to awaken the waiting thread. The `strat` function reports an error to `physio` by setting B_ERROR in `bp->b_flags` and putting an errno value in `bp->b_error`. See the errno values in `<sys/errno.h>`.

The `bp` parameter is a pointer to a `buf` structure allocated by the caller. If set to NULL, `physio` allocates the `buf` structure for the caller. The `physio` function deallocates the `buf` structure when the I/O request completes.

The `dev` parameter is the device number passed to the driver.

The `flag` parameter indicates the direction of the I/O transfer. B_READ is used to read data from the device into memory; B_WRITE is used to write data from memory to the device.

The `mincnt` parameter is a pointer to a function that limits the data transfer length. Typically, drivers pass the `minphys` kernel supplied function as this parameter.

The `uiop` parameter is a pointer to the `uio` structure that is passed to the driver. The `uio` structure specifies the following:

- **uio_iov**: Pointer to an `iovec` structure that contains the base address (`iov_base`) and transfer length (`iov_len`) of the I/O request.
- **uio_iovcnt**: Number of `iovec` structures. If >1, `uio_iov` points to an array of `iovec` structures.
- **uio_offset**: Offset into device.
**uio_seg**    Type of memory segment to transfer. If set to UIOSEG_USER, physio must be called in the user context.

**uio_resid** Number of bytes of data remaining to be transferred.

For each iovec structure, physio performs the following:

1. Validates the user has appropriate access permissions for the data pages specified by iovec.
2. Sets up the buf structure with the following information:
   - *b_dev* is set to the device number.
   - *b_error* is set to zero.
   - *b_flags* is set with B_BUSY, B_PHYS, and B_RAW. If the flag parameter is B_WRITE, B_WRITE is set.
   - *b_un.b_addr* is set to *iov_base*.
   - *b_bcount* is set to *iov_len*. This value can be adjusted by *mincnt*.
   - *b_blkno* is set to the DEV_BSIZE block number corresponding to *uio_offset*.
3. Calls *mincnt* to adjust the transfer length, if too large. If adjusted, physio will make multiple calls to *strat* until all the data specified by *iov_len* has been transformed (or an error occurs).
4. If *uio_seg* is not UIOSEG_KERNEL, locks down the data pages to be transferred.
5. If *uio_seg* is not UIOSEG_KERNEL and the driver has set C_MAP_BUFFER_TO_KERNEL in the *d_flags* field of its *drv_ops_t* structure, physio maps the user buffer into kernel space. This additional overhead is performed for legacy drivers that do not know how to access user space in their strategy function.
6. Calls *strat* passing the buf structure pointer *bp* as a parameter, then waits for the I/O request to complete by calling *biowait*.
7. The driver calls *biodone* when the I/O request completes to awaken the thread waiting in *biowait*.
8. After the I/O request has completed, data pages that have been locked down are unlocked.
9. The following fields in the *uio* structure are updated:
   - *uio_resid* is decreased by the transfer length.
   - *uio_offset* is increased by the transfer length.

After all iovec structures have been processed, the buf structure is released and physio returns to the caller.

**RETURN VALUES**

0 Successful completion.
<>0 Error.

**CONSTRAINTS**

Do not call in an interrupt context.
Do not call while holding a spinlock.

**EXAMPLE**

```c
int mydriver_read(dev_t dev, struct uio * uio)
{
    return physio(mydriver_strategy, NULL, dev, B_READ,
                  minphys, uio);
}
```

**SEE ALSO**

biodone(9F), biowait(9F), buf(9F), minphys(9F), uio(9F)
printf(9F)

NAME

printf - Kernel print routine.

SYNOPSIS

#include <sys/kern_svcs.h>

int printf(
    const char *fmt,
    ...
);

PARAMETERS

fmt  A set of printing characters and limited conversion specifications, as defined in printf(3S).
...  An argument corresponding to a format conversion specification, as defined in printf(3S).

DESCRIPTION

The printf kernel function is a scaled down version of the C library printf routine (see printf(3S)).

The printf kernel function writes diagnostic information to the console and into the msgbuf kernel message buffer.

The printf kernel routine can accept the following formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Printed Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>%%</td>
<td>%; no argument</td>
</tr>
<tr>
<td>%b</td>
<td>Characters from string argument; can include backslash-escape codes</td>
</tr>
<tr>
<td>%c</td>
<td>Character from integer argument</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal from integer argument</td>
</tr>
<tr>
<td>%lx</td>
<td>The argument x is a long integer; x is one of x,d,u, or o</td>
</tr>
<tr>
<td>%o</td>
<td>Octal from integer argument</td>
</tr>
<tr>
<td>%s</td>
<td>Characters from string argument</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal from integer argument</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal from integer argument</td>
</tr>
</tbody>
</table>

These formats are the same as in printf(3S). Other formats specified in printf(3S) are not supported.

The printf kernel function is buffered. Therefore, console error messages may not be synchronous with the current kernel state.

RETURN VALUES

The printf kernel function returns the length of the formatted string.

CONSTRAINTS

None

SEE ALSO

vprintf(9F), vsnprintf(9F), msg_printf(9F), printf(3S)
NAME

priv_policy - Determine if the credential has a privilege asserted.

SYNOPSIS

#include <sys/privileges.h>
#include <sys/priv.h>

int priv_policy(
    const cred_t *cr,
    priv_t priv,
    int err,
    const char *msg
);  

PARAMETERS

cr    Credential to be checked.
priv  Privilege for which to check assertion.
err   Error code to return.
msg   Debug message.

DESCRIPTION

The priv_policy function determines whether the credential has the privilege priv asserted. A privilege can be asserted in the following ways:

• The privilege is in the effective privilege set of the credential.
• The thread has an effective uid equal to 0, priv is part of a root replacement privilege, and if compartments are enabled, priv is not disabled in the thread’s compartment.
• The privilege is granted to all processes (not implemented).

RETURN VALUES

0    Success. Privilege is asserted.
err  Failure. Privilege is not asserted or credential is null.

NOTES

The priv_policy function is similar to privileged_cred, except that it does the following:

• Returns 0 on success; privileged_cred returns 1 on success.
• Returns a third argument on failure; privileged_cred returns 0 on failure.
• Accepts a debug message as fourth argument.

The following call:
privileged_cred(CRED, PRIV_FOO)
Is equivalent to:
!priv_policy(CRED, PRIV_FOO, 1, """)

CONSTRAINTS

Call this function from the user context.

The priv_policy function can call printf to generate diagnostic messages. The printf function uses a spinlock of order PRINTF_LOCK_ORDER. Therefore, do not call these functions if holding a spinlock of order PRINTF_LOCK_ORDER or higher.
EXAMPLES

The following code example checks for `drv_priv` to unlock a BTLAN driver request:

```c
if (drv_priv(ioc_reqp->ioc_cr) != 0) {
    /* not super user */
    status = EPERM;
    break;
}
```

The following code example shows the preceding code fragment with changes to use `priv_policy`:

```c
if (priv_policy(ioc_reqp->ioc_cr, PRIV_NETADMIN, EPERM, "btlan_ctl_req") == EPERM) {
    /* unprivileged user */
    status = EPERM;
    break;
}
```

WARNINGS

None

SEE ALSO

`acct(4), compartments(5), privileges(3), privileges(5), priv_scall(9F)`. 
priv_scall(9F)

NAME

priv_scall - Determine if the calling thread has a given privilege asserted.

SYNOPSIS

#include <sys/privileges.h>
#include <sys/priv.h>

int priv_scall(
    priv_t priv,
    const char *msg
);

PARAMETERS

priv    Privilege for which to check assertion.
msg     Debug message.

DESCRIPTION

The priv_scall function determines whether the calling thread (kthread_t) has the privilege priv asserted. A privilege can be asserted in the following ways:
• The privilege is in the effective privilege set of the thread.
• The thread has an effective uid equal to 0, priv is part of a root replacement privilege, and if compartments are enabled, priv is not disabled in the thread’s compartment.
• The privilege is granted to all processes (not implemented).

The function also adjusts some accounting information of the calling thread. In particular, it sets the ASU flag if the routine determines that the priv is considered to be asserted as a side effect of caller’s euid being 0.

RETURN VALUES

0  Failure. Privilege is not enabled.
1  Success. Privilege is enabled.

NOTES

Unlike suser, priv_scall does not explicitly set u.u_error to EPERM when privilege is not asserted. The code must have a mechanism to return the EPERM value to the caller.

The rest of the kernel code must not check whether a thread of credential represents rootuser (that is, has an euid of 0) using a different mechanism, such as CR_EUID.

CONSTRAINTS

Call this function from the user context.

The priv_scall function can call printf to generate diagnostic messages. The printf function uses a spinlock of order PRINTF_LOCK_ORDER. Therefore, do not call these functions if holding a spinlock of order PRINTF_LOCK_ORDER or higher.

EXAMPLES

The following code example uses the obsolete suser function to determine whether the calling thread is privileged:

if (!suser()) {
    return EPERM;
}
The following code example shows the preceding code fragment with changes to use `priv_scall`:
#include <sys/privileges.h>

if (!priv_scall(PRIV_NETADMIN, "net diag ioctl")) {
    return EPERM;    /* Return status to caller */
}

WARNINGS
None

SEE ALSO
privileged_cred(9F)

NAME

privileged_cred - Determine if a credential has a specified privilege asserted.

SYNOPSIS

#include <sys/privileges.h>
#include <sys/priv.h>

int privileged_cred(
    const cred_t *cr,
    priv_t priv
);

PARAMETERS

cr    Credential to be checked.
priv  Privilege for which to check whether it is asserted.

DESCRIPTION

The privileged_cred function determines whether the credential has the privilege priv asserted. A privilege can be asserted in the following ways:

• The privilege is in the effective privilege set of the thread.
• The thread has an effective uid equal to 0, priv is part of a root replacement privilege, and if compartments are enabled, priv is not disabled in the thread’s compartment.
• The privilege is granted to all processes (not implemented).

RETURN VALUES

The privileged_cred function returns the following values:

0  Failure. The privilege is not enabled or the credential is null.
1  Success. The privilege is enabled.

NOTES

The privileged_cred function does not update the ASU bit and does not generate audit information.

LOCKING CONSIDERATIONS

The privileged_cred function can call printf to generate diagnostic messages. The printf function uses a spinlock of order PRINTF_LOCK_ORDER. This means that the callers of these functions must not hold a spinlock of order PRINTF_LOCK_ORDER or higher.

EXAMPLES

The following example uses drv_priv to unlock a BTLAN driver request:

if (drv_priv(ioc_reqp->ioc_cr) != 0) {
    /* not superuser */
    status = EPERM;
    break;
}

The following example uses privileged_cred to unlock a BTLAN driver request:

if (!privileged_cred(ioc_reqp->ioc_cr, PRIV_NETADMIN)) {
    /* unprivileged user */
    status = EPERM;
    break;
}
SEE ALSO

NAME
privlbcopy - Copy data from a source buffer to a destination buffer using global virtual addresses.

SYNOPSIS
#include <sys/kern_svc.s.h>
int privlbcopy(
    space_t from_sid,  
caddr_t from_addr,  
    space_t to_sid,  
caddr_t to_addr,  
    size_t n
);  

PARAMETERS
from_sid  Space ID of source buffer.
from_addr Address of source buffer.
to_sid  Space ID of destination buffer.
to_addr Address of destination buffer.
n  Number of bytes to copy.

DESCRIPTION
The privlbcopy kernel function copies data from a source buffer to a destination buffer using
global virtual addresses. Protection is disabled to allow copies from/to buffers that are not owned
by the current process.
The from_sid and from_addr parameters are the space ID and address, respectively, of the source
buffer.
The to_sid and to_addr parameters are the space ID and address, respectively, of the destination
buffer.
The space ID (SID) and address are combined to form the global virtual address of the buffer. If
the address is in kernel space, use ldsid to get the SID of the address. Do not call privlbcopy
to copy data from or to user space without first calling physio. The physio function validates
access to the user buffer, locks the corresponding data pages in memory, puts the user SID and
address in the b_spaddr and b_un.b_addr fields of the buf structure. (b_spaddr and
b_un.b_addr will contain a kernel space ID and address if the driver specifies
C_MAP_BUFFER_TO_KERNEL in its drv_ops_t structure.)

RETURN VALUES
0     Successful completion.
<>0  Error.

CONSTRAINTS
None

EXAMPLE
/*
 * Copy data from a kernel buffer to a user buffer that
 * has been validated and locked by physio().
 */
(void)privlbcopy(ldsid(kern_addr), kern_addr,
    bp->b_spaddr, bp->b_un.b_addr, bp->b_bcount);

**WARNING**

Do not use privlbcopy for transfers between memory and I/O space. The underlying routines make choices of the optimal transfer code, which will probably not be supported on either the I/O bus adapter or the attached interface card. Failure to heed this warning may result in data corruption, a High Priority Machine Check (HPMC), or a call to panic.

**SEE ALSO**

`bcopy(9F), copyin(9F), copyout(9F), ldsid(9F), physio(9F)`
**proc_pid_self(9F)**

**NAME**

proc_pid_self - Get the process ID of the calling process.

**SYNOPSIS**

```c
#include <sys/types.h>
#include <sys/proc_id.h>
pid_t proc_pid_self(void);
```

**PARAMETERS**

None

**DESCRIPTION**

The `proc_pid_self` function obtains the process ID of the calling process.

**RETURN VALUES**

`n` A nonnegative process ID.

**CONSTRAINTS**

Do not call on the ICS.

**EXAMPLES**

```c
#include <sys/types.h>
#include <sys/proc_id.h>

/*
 * Example : Get the pid of the calling process
 */

{
    pid_t pid;
    pid = proc_pid_self();
}
```

**SEE ALSO**

None
NAME

procp_self - Get a pointer to a proc structure for a calling process

SYNOPSIS

#include <sys/proc.h>

proc_t *procp_self(void);

DESCRIPTION

The proc structure (struct proc or proc_t) is a core kernel private structure of indeterminate size.

RETURN VALUES

The procp_self routine returns a pointer to the proc structure associated with the calling process.

CONSTRAINTS

None

SEE ALSO

kthreadp_self(9F)
NAME
psignal - Send the specified signal to a process.

SYNOPSIS
#include <sys/proc_iface.h>

void psignal(
    proc_t *procp,
    int sig
);

PARAMETERS
procp  Pointer to a proc_t structure.
sig    Signal number.

DESCRIPTION
The psignal kernel function sends the specified sig to the process specified by procp.
The procp parameter is a pointer to a proc_t structure. Use the procp_self routine to obtain
the pointer to the proc structure for the current process.
Signal numbers are defined in <sys/signal.h>.
The psignal function checks the signal mask of the process to determine if the specified signal
is being blocked, ignored, or caught. If a process is blocking the signal, the signal is recorded as
pending so the signal is not lost. If a process is ignoring the signal, the signal is not sent. If the
process is catching the signal, that process is put on the run queue. For example, if the process
has called sleep with priority greater than PZERO and PCATCH set, psignal will awaken the
sleeping process.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
#include <sys/proc.h>
#include <sys/signal.h>
#include <sys/proc_iface.h>

    
    /*
    * In the user context, save the pointer to the proc structure.
    */
    proc_t *my_procp = procp_self();
    ...

    /*
    * In the ISR, signal the process.
    */
    psignal(my_procp, SIGIO);

SEE ALSO
gsignal(9F)
NAME
putc - Manipulate characters on a clist.

SYNOPSIS
#include <sys/clist.h>
int putc(
    int ch,
    struct clist *list
);

PARAMETERS
ch    Character to place on the clist.
list   Pointer to a clist.

DESCRIPTION
The putc kernel function puts the character ch on the clist list.

RETURN VALUES
0   Successful completion.
-1  Error (probably exhausted the available list structures).

CONSTRAINTS
None

EXAMPLE
void catq( from, to )
struct clist * from;
struct clist * to;

/* concatenate characters from one clist to another */
/* expects putc to succeed */
{
    int c;

    while ((c = getc(from)) >= 0)
        if(putc(c, to))

SEE ALSO
getc(9F), getb(9F), getcf(9F), putb(9F), putcf(9F)
NAME
putcb - Manipulate cblocks on a clist.

SYNOPSIS
#include <sys/clist.h>
void putcb(
    struct cblock *cb,
    struct clist *list
);

PARAMETERS
cb Pointer to a cblock.
list Pointer to a clist.

DESCRIPTION
The putcb kernel function adds the cblock pointed to by cb onto the clist pointed to by list.

RETURN VALUES
None

CONSTRAINTS
None

SEE ALSO
getcb(9F), getc(9F), getcf(9F), putc(9F), putcf(9F)
NAME
putcf - Manipulate a cblock on the cblock free list.

SYNOPSIS
#include <sys/clist.h>
void putcf(
    struct cblock *cb
);

PARAMETERS
cb Pointer to a cblock structure.

DESCRIPTION
The `putcf` kernel function returns the cblock pointed to by `cb` to the free list.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
static int
mydev_open(dev_t dev, int flag)
{
    struct cblock *cp;
    struct mydevice dv;

    /* Flush out the device buffer */
    while ((cp = getcb(&dv->queue)) != NULL)
        putcf(cp);

    ...
}

SEE ALSO
getc(9F), getc(9F), getcb(9F), putc(9F), putcb(9F)
**NAME**

rwlock_alloc - Dynamically allocate and initialize a reader-writer lock

**SYNOPSIS**

```c
#include <sys/rwlock.h>

rwlock_t rwlock_alloc(
    char *name,
    rwlock_attr_t *attr,
    rwlock_attr_flg_t alloc_flag,
    unsigned int major_order,
    uintptr_t minor_order);
```

**PARAMETERS**

- `name` The reader-writer lock name.
- `attr` Pointer to a reader-writer lock attribute (`rwlock_attr_t`) structure or NULL.
- `alloc_flag` `RWL_WAITOK` or `RWL_NOWAIT`; these are mutually exclusive.
- `major_order` The major order number that is used for deadlock detection in a debug kernel.
- `minor_order` The order for locks with the same major order number. Frequently, the address of the structure being protected.

**DESCRIPTION**

The `rwlock_alloc` kernel function dynamically allocates and initializes a reader-writer lock. Reader-writer locks support the following attributes:

- `RWL_ATTR_PRIOCEIL` The reader-writer lock participates in the priority ceiling protocol.
- `RWL_ATTR_NOSWAP` The thread holding the lock will not be deactivated.
- `RWL_ATTR_DISOWNABLE` The reader-writer lock can be disowned.
- `RWL_ATTRSAME_ORDER_OK` This corresponds to the deadlock safe feature of HP-UX 11i v2 rwlocks. The thread can hold other locks of the same major and minor order.

In the future, additional flags might be supported that provide performance hints to the reader-writer lock routines.

**RETURN VALUES**

- `<>NULL` Pointer to the allocated reader-writer lock.
- `NULL` The allocation failed because of a lack of memory (in the case of `RWL_NOWAIT`).

**CONSTRAINTS**

Must be called in a thread context and with no spinlocks held, if `RWL_WAITOK` is specified.

If `attr` is a pointer, the attribute structure must be initialized by `rwlock_attr_init` and `rwlock_attr_setflag` before calling this routine.

**EXAMPLE**

```c
#include <sys/rwlock.h>

rwlock_t *count_lock_p;
```
/ * Use the reader-writer lock. * /
count_lock_p = rwlock_alloc ("My RWlock Lock", &my_attr, RWL_WAITOK,
       MY_MAJOR_NUMBER, NULL);

/*
 * Done with the lock.
 */
rwlock_dealloc (count_lock_p);

SEE ALSO
rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock_downgrade(9F),
rwlock_owned(9F), rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_tryupgrade(9F), rwlock_trywrlock(9F),
rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F), rwlock_wrowned(9F)
rwlock_attr_init(9F)

NAME
rwlock_attr_init - Initialize optional reader-writer lock attributes

SYNOPSIS
#include <sys/rwlock.h>
void rwlock_attr_init(
    rwlock_attr_t *attr,
    size_t size
);

PARAMETERS
attr A pointer to the reader-writer attribute (rwlock_attr_t) structure.
size The size of the reader-writer lock attribute structure.

DESCRIPTION
The rwlock_attr_init routine initializes the reader-writer lock attribute structure. Kernel
modules can pass optional attributes to the reader-writer creation routines through an attribute
data structure. Before using this structure, kernel modules must initialize the structure to its
default values. After the data structure is initialized, the module can set the specific attributes
for the reader-writer lock with rwlock_attr_setflag and then call the creation routine.

Reader-writer locks support the following attributes:

R WL _ ATT R _ P RIO CEIL The reader-writer lock participates in the priority ceiling
protocol.
R WL _ ATT R _ NO S WAP The thread holding the lock will not be deactivated.
R WL _ ATT R _ D ISOWNABLE The reader-writer lock can be disowned.
R WL _ ATT R _ S A ME _ O RDER _ OK This corresponds to the deadlock safe feature of HP-UX 11i
v2 rwlocks. The thread can hold other locks of the same major
and minor order.

The default value for the attribute structure is none of the above. After the attribute structure is
initialized to its default value, use the rwlock_attr_setflag to set specific flags in the attribute
structure.

In the future, additional flags might be supported that provide performance hints to the
reader-writer lock routines.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
#include <sys/rwlock.h>

rwlock_t *count_lock_p;
rwlock_attr_t my_attr;
/*
 * Set default values.
 */
rwlock_attr_init(&my_attr, sizeof(rwlock_attr_t));
/*
* Turn on priority ceiling and same order okay.
  */
  rwlock_attr_setflag(&my_attr, RWL_ATTR_PRIOCeil | RWL_ATTR_SAME_ORDER_OK,
                      RWL_ATTR_FLAG_ON);
  count_lock_p = rwlock_alloc("My rwlock Lock", &my_attr, RWL_WAITOK,
                               MY_MAJOR_NUMBER, NULL);

SEE ALSO

rwlock_alloc(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock_downgrade(9F), rwlock_owned(9F),
rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_tryupgrade(9F), rwlock_trywrlock(9F),
rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F), rwlock_wrowned(9F)
NAME

rwlock_attr_setflag – Set reader-writer lock attribute structure values.

SYNOPSIS

#include <sys/rwlock.h>

void rwlock_attr_setflag(
    rwlock_attr_t *attr,
    rwlock_attr_flg_t flags,
    rwlock_attr_flg_set_t mode
);

PARAMETERS

attr      A pointer to the reader-writer attribute (rwlock_attr_t) structure.
flags     The attributes to turn on or off. This is logical OR of the attributes in the DESCRIPTION section.
mode      The attribute’s mode: RWL_ATTR_FLAG_ON (on) or RWL_ATTR_FLAG_OFF (off).

DESCRIPTION

The rwlock_attr_setflag routine sets the specified attributes on or off in the reader-writer lock attribute structure. Kernel modules can pass optional attributes to the reader-writer creation routines through an attribute data structure. Before using this structure, kernel modules must initialize the structure to its default values. After the data structure is initialized, the module uses rwlock_attr_setflag to set the specific attributes for the reader-writer lock. Then, the attribute structure is passed to the creation routine.

Reader-writer locks support the following attributes:

- RWL_ATTR_PRIOCEIL: The reader-writer lock participates in the priority ceiling protocol.
- RWL_ATTR_NOSWAP: The thread holding the lock will not be deactivated.
- RWL_ATTR_DISOWNABLE: The reader-writer lock can be disowned.
- RWL_ATTRSAME_ORDER_OK: This corresponds to the deadlock safe feature of HP-UX 11i v2 rwlocks. The thread can hold other locks of the same major and minor order.

After the attribute structure is initialized to its default value, use the rwlock_attr_setflag to set specific flags in the attribute structure.

In the future, additional flags might be supported that provide performance hints to the reader-writer lock routines.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

#include <sys/rwlock.h>

rwlock_t *count_lock_p;
rwlock_attr_t my_attr; /*
* Set default values.
  */
rwlock_attr_init(&my_attr, sizeof(rwlock_attr_t));
  /*
  * Turn on priority ceiling and same order okay.
  */
rwlock_attr_setflag(&my_attr, RWL_ATTR_PrioCeil | RWL_ATTR_SAME_ORDER_OK,
                    RWL_ATTR_FLAG_ON);

count_lock_p = rwlock_alloc("My rwlock Lock", &my_attr, RWL_WAITOK,
                         MY_MAJOR_NUMBER, NULL);

SEE ALSO

rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_dealloc(9F), rwlock_downgrade(9F), rwlock_owned(9F),
rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_tryupgrade(9F), rwlock_trywrlock(9F),
rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F), rwlock_wrowned(9F)
**NAME**

`rwlock_dealloc` - Destroy and deallocate a reader-writer lock

**SYNOPSIS**

```c
#include <sys/rwlock.h>

void rwlock_dealloc(  
    rwlock_t *rwlp
);
```

**PARAMETERS**

`rwlp`  
Pointer to a reader-writer lock.

**DESCRIPTION**

The `rwlock_dealloc` kernel function destroys and deallocates a reader-writer lock.

**RETURN VALUES**

None

**CONSTRAINTS**

Must be called with the reader-writer lock unlocked.

**EXAMPLE**

```c
#include <sys/rwlock.h>

rwlock_t *count_lock_p;

/*
 * Use the reader-writer lock.
 */
count_lock_p = rwlock_alloc ("My RWlock Lock", &my_attr, RWL_WAITOK,
    MY_MAJOR_NUMBER, NULL);

/*
 * Can done with the lock.
 */
rwlock_dealloc (count_lock_p);
```

**SEE ALSO**

`rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_downgrade(9F), rwlock_owned(9F), rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_tryupgrade(9F), rwlock_trywrlock(9F), rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F), rwlock_wrowned(9F)`
NAME

rwlock_downgrade - Downgrade a lock held for write access

SYNOPSIS

#include <sys/rwlock.h>

void rwlock_downgrade(
    rwlock_t *rwlp
);

PARAMETERS

rwlp    Pointer to a reader-writer lock.

DESCRIPTION

The rwlock_downgrade kernel function downgrades a lock held for write access to read access. The caller retains access to the lock. No intervening writers are given access to the lock. If there are readers at the head of the wait queue, they are also granted read access.

RETURN VALUES

None

CONSTRAINTS

Must be called in a process context.

EXAMPLE

#include <sys/rwlock.h>

rwlock_t *lock_p;
struct data {int count, user_data;};
struct data my_data, snapshot_of_data;

int try_update_my_data ( int new_data)
{
    /*
     * Have read access upon entry
     */
    if (!rwlock_tryupgrade (lock_p)
        return (FAILED);
    /*
     * Return with read access still held
     */
    /*
     * Update the data now that we have write access.
     */
    my_data.count ++;
    my_data.user_data = new_data;
    /*
     * Return with read access still held
     */
    rwlock downgrade (lock_p);
}

SEE ALSO

rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock_owned(9F), rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_trygrade(9F), rwlock_trywrlock(9F), rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_kwrl0ck(9F), rwlock_kwrowned(9F)
rwlock_owned(9F)

NAME
rwlock_owned - Return current lock mode for calling thread

SYNOPSIS
#include <sys/rwlock.h>
rwlock_owned_t rwlock_owned(
    rwlock_t *rwlp
);

PARAMETERS
rwlp Pointer to a reader-writer lock.

DESCRIPTION
The rwlock_owned kernel function returns the current lock mode for the calling thread. The possible modes are the thread does not own the lock, the thread has read access, or the thread has write access.

Because of the overhead of maintaining this information, this routine is only available in debug kernels and is intended for use in asserts.

RETURN VALUES
RWL_READLOCKED The thread owns the lock for read access.
RWL_WRITELOCKED The thread owns the lock for write access.
RWL_REC_READLOCKED The thread owns the lock for recursive read access.
RWL_REC_WRITELOCKED The thread owns the lock for recursive write access.
RWL_UNLOCKED Not locked by the current thread.

CONSTRAINTS
None

EXAMPLE
#include <sys/rwlock.h>
rwlock_t *lock_p;
struct data {int count, user_data;};
struct data my_data;
/*
 * Better have write access; we are going to update my_data.
 */
VASSERT ((rwlock_owned (lock_p) == RWL_WRITELOCKED) ||
    (rwlock_owned(lock_p) == RWL_REC_WRITELOCKED));
my_data.count ++;
my_data.user_data = 100;

SEE ALSO
rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock_downgrade(9F),
rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_tryupgrade(9F), rwlock_trywrlock(9F),
rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F), rwlock_wrowned(9F),
NAME

rwlock_rdlock - Acquire a read lock on a reader-writer lock

SYNOPSIS

#include <sys/rwlock.h>

void rwlock_rdlock(
    rwlock_t *rwlp
);

PARAMETERS

rwlp Pointer to a reader-writer lock.

DESCRIPTION

The rwlock_rdlock kernel function acquires a read lock on a reader-writer lock. If writers are waiting for write access, read access is not granted until all of the waiting writers have been serviced. If the thread blocks waiting for access to the lock, it is not awakened by a signal.

RETURN VALUES

None

CONSTRAINTS

Must be called only from a thread context. The thread can neither hold any spinlocks nor already hold the specified lock.

EXAMPLE

#include <sys/rwlock.h>

rwlock_t *count_lock_p;
header_t header;

data_t *search_list(int val)
{
    data_t *next;
    /* Search the list for val.
    * The list is protected by lock_p.
    */
    rwlock_rdlock(lock_p);
    next = header.head_of_list;
    while ((next!= NULL) && (next-> != val))
    {
        next = next->next;
        /*
        * Release the lock when done.
        */
        rwlock_unlock(lock_p);
        return (next);
    }
}

SEE ALSO

rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock_downgrade(9F), rwlock_owned(9F), rwlock_tryrdlock(9F), rwlock_tryupgrade(9F), rwlock_trywrlock(9F), rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F), rwlock_wrowned(9F)
**rwlock_tryrdlock(9F)**

**NAME**

rwlock_tryrdlock - Conditionally acquire a read lock on a reader-writer lock

**SYNOPSIS**

```c
#include <sys/rwlock.h>

int rwlock_tryrdlock(
    rwlock_t *rwlp
);
```

**PARAMETERS**

- `rwlp` Pointer to a reader-writer lock.

**DESCRIPTION**

The `rwlock_tryrdlock` kernel function conditionally acquires a read lock on a reader-writer lock.

**RETURN VALUES**

- `<0` Obtained read access.
- `0` The `rwlock_tryrdlock` function failed to obtain read access.

**CONSTRAINTS**

Can be called in an interrupt context.

The calling thread cannot already hold the specified lock.

**EXAMPLE**

```c
#include <sys/rwlock.h>

rwlock_t *lock_p;
header_t header;

data_t *search_list (int flags, int val)
{
    data_t *next;
    if (flags == NOWAIT) {
        /* Cannot wait for the lock */
        if ( !rwlock_tryrdlock (lock_p)) {
            return (FAILED_TO_DO_WORK);
        }
    }
    else rwlock_rdlock(lock_p);
    /* Search the list for val.
     * The list is protected by lock_p
     */
    next = header.head_of_list;
    while ( (next !=NULL) && (next->val != val))
        next = next->next;
    rwlock_unlock (lock_p);
    return (next);
}
```

**SEE ALSO**

rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock_downgrade(9F), rwlock_owned(9F), rwlock_rdlock(9F), rwlock_tryupgrade(9F), rwlock_trywrlock(9F), rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F), rwlock_wrowned(9F)
NAME

rwlock_tryupgrade - Conditionally upgrade a lock held for read access

SYNOPSIS

#include <sys/rwlock.h>

int rwlock_tryupgrade(
    rwlock_t *rwlp
);

PARAMETERS

rwlp  Pointer to a reader-writer lock.

DESCRIPTION

The rwlock_tryupgrade kernel function upgrades a lock held for read access to write access if there are no other readers and no waiting writers.

The current read lock must have been acquired by a non-recursive lock routine (for example, rwlock_rdlock or rwlock_tryrdlock).

RETURN VALUES

<>0  Success.

0  Failure.

CONSTRAINTS

Must be called in a process context.

EXAMPLE

#include <sys/rwlock.h>

rwlock_t *lock_p;
struct data {int count, user_data;};
struct data my_data, snapshot_of_data;

int try_update_my_data ( int new_data)
{
    /*
     * Have read access upon entry
     */
    if (!rwlock_tryupgrade (lock_p)
        return (FAILED);
    /*
     * Return with read access still held
     */
    /*
     * Update the data now that we have write access.
     */
    my_data.count ++;
    my_data.user_data = new_data;
    /*
     * Return with read access still held
     */
    rwlock_downgrade (lock_p);
}
SEE ALSO

rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock downgrade(9F), rwlock_owned(9F), rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_trywrlock(9F), rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F), rwlock_wrowned(9F)
**rwlock_trywrlock(9F)**

**NAME**

rwlock_trywrlock - Conditionally acquire a write lock on a reader-writer lock

**SYNOPSIS**

```c
#include <sys/rwlock.h>

int rwlock_trywrlock(
    rwlock_t *rwlp
);
```

**PARAMETERS**

- **rwlp** Pointer to a reader-writer lock.

**DESCRIPTION**

The `rwlock_trywrlock` kernel function conditionally acquires a write lock on a reader-writer lock.

**RETURN VALUES**

- `<0` Obtained write access.
- `0` The `rwlock_trywrlock` function failed to obtain write access.

**CONSTRAINTS**

Can be called in an interrupt context.
The calling thread cannot already hold the specified lock.

**EXAMPLE**

```c
#include <sys/rwlock.h>

rwlock_t *lock_p;
header_t header;

int add_to_list (int flags, data_t *data)
{
    if (flags == NOWAIT) {
        /* Cannot wait for the lock */
        if ( !rwlock_trywrlock (lock_p)) {
            return (FAILED_TO_DO_WORK);
        }
    }
    else rwlock_wrlock(lock_p);
    /* Add to head of singly linked list.
    * The list is protected by lock_p
    */
    data->next = header.head_of_list;
    header.head_of_list = data;
    rwlock_unlock (lock_p);
    return (OK);
}
```

**SEE ALSO**

`rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock_downgrade(9F), rwlock_owned(9F), rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_tryupgrade(9F), rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F), rwlock_wrowned(9F)`
NAME
rwlock_unlock - Unlock a reader-writer lock

SYNOPSIS
#include <sys/rwlock.h>
void rwlock_unlock(
    rwlock_t *rwlp
);

PARAMETERS
rwlp    Pointer to a reader-writer lock.

DESCRIPTION
The rwlock_unlock kernel function unlocks a reader-writer lock, waking up waiting threads if their request can be granted. Waiting threads are unblocked based on their arrival time (FIFO).

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
#include <sys/rwlock.h>

rwlock_t *count_lock_p;
header_t header;

int add_to_list(data_t *data)
{
    /* Add to head of single linked list.
     * The list is protected by lock_p.
     */
    rwlock_wrlock(lock_p);
    data->next = header.head_of_list;
    header.head_of_list = data;
    /*
     * Release the lock when done.
     */
    rwlock_unlock(lock_p);
    return (OK);
}

SEE ALSO
rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock_downgrade(9F), rwlock_owned(9F), rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_tryupgrade(9F), rwlock_trywrlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F), rwlock_wrowned(9F)
NAME

rwlock_upgrade - Upgrade a lock held for read access

SYNOPSIS

#include <sys/rwlock.h>

void rwlock_upgrade(
    rwlock_t *rwlp
);

PARAMETERS

rwlp Pointer to a reader-writer lock.

DESCRIPTION

The rwlock_upgrade kernel function upgrades a lock held for read access to write access. If
this routine succeeds, the caller is guaranteed that no intervening writer has been granted the
lock. This routine might block until all reads have completed. This routine fails if there is a
pending upgrade request. If this routine fails, the thread retains its read access.

RETURN VALUES

<>0 Success.
0 Failure.

CONSTRAINTS

Must be called from a thread context.
The calling thread cannot already hold spinlocks.

EXAMPLE

#include <sys/rwlock.h>

rwlock_t *lock_p;
struct data {int count, user_data;};
struct data my_data, snapshot_of_data;

void copy_and_update_my_data (int new_data)
{
    /*
     * My_data is protected by lock_p. The copy of my_data
     * to snapshot_of_my_data and the update of my_data need
     * to be in sync
     */
    rwlock_rdlock (lock_p);
    snapshot_of_data = my_data;
    /*
     * Copy the data
     */
    if (!rwlock_upgrade (lock_p) { /*
        * Release read lock, get write lock, and recopy the data.
        */
        rwlock_unlock (lock_p);
        rwlock_wrlock (lock_p);
        /*
        * Snapshot is now stale, need to recopy it
        */
        snapshot_of_data = my_data;
} */
    * Update the data now that we have write access.
    */
    my_data.count ++;
    my_data.user_data = new_data;
    rwlock_unlock (lock_p);
}

SEE ALSO

rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock_downgrade(9F),
rwlock_owned(9F), rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_tryupgrade(9F), rwlock_trywrlock(9F),
rwlock_unlock(9F), rwlock_wrlock(9F), rwlock_wrowned(9F)
**NAME**

rwlock_wrlock - Acquire a write lock on a reader-writer lock

**SYNOPSIS**

```c
#include <sys/rwlock.h>

void rwlock_wrlock(
    rwlock_t *rwlp
);
```

**PARAMETERS**

- `rwlp` Pointer to a reader-writer lock.

**DESCRIPTION**

The `rwlock_wrlock` kernel function acquires a write lock on a reader-writer lock. If necessary, the calling thread blocks until the reader-writer lock is available. If the thread blocks, it is not awakened by a signal.

**RETURN VALUES**

None

**CONSTRAINTS**

Must be called only from a thread context.

The thread can neither hold any spinlocks nor already hold the specified lock.

**EXAMPLE**

```c
#include <sys/rwlock.h>

rwlock_t *count_lock_p;
header_t header;

int add_to_list(data_t *data)
{
    /* Add to head of single linked list. 
       * The list is protected by lock_p. 
       */
    rwlock_wrlock(lock_p);
    data->next = header.head_of_list;
    header.head_of_list = data;
    /*
       * Release the lock when done.
       */
    rwlock_unlock(lock_p);
    return (OK);
}
```

**SEE ALSO**

- `rwlock_alloc(9F)`, `rwlock_attr_init(9F)`, `rwlock_attr_setflag(9F)`, `rwlock_dealloc(9F)`, `rwlock_downgrade(9F)`,
- `rwlock_owned(9F)`, `rwlock_rdlock(9F)`, `rwlock_tryrdlock(9F)`, `rwlock_tryupgrade(9F)`, `rwlock_trywrlock(9F)`,
- `rwlock_unlock(9F)`, `rwlock_upgrade(9F)`, `rwlock_wrowned(9F)`
NAME
rwlock_wrowned - Tests whether the caller has a write lock on a reader-writer lock

SYNOPSIS
#include <sys/rwlock.h>

int rwlock_wrowned(
    rwlock_t *rwlp
);

PARAMETERS
rwlp Pointer to a reader-writer lock.

DESCRIPTION
The rwlock_wrowned kernel function tests whether the caller has a recursive write lock on the
lock. This function is available in a perf kernel. Because of the overhead involved, you can only
determine if the thread as read access in a debug kernel (see rwlock_owned(9F)).

RETURN VALUES
<>0 The thread owns the lock for write access.
0 The thread does not have write access.

CONSTRAINTS
None

EXAMPLE
#include <sys/rwlock.h>

rwlock_t *lock_p;
struct data {int count, user_data;};
struct data my_data;

VASSERT (rwlock_wrowned (count_lock_p));
    /*
     * Better have write access; we are going to update my_data.
     */
    my_data.count ++;
    my_data.user_data = 100;

SEE ALSO
rwlock_alloc(9F), rwlock_attr_init(9F), rwlock_attr_setflag(9F), rwlock_dealloc(9F), rwlock_downgrade(9F),
rwlock_owned(9F), rwlock_rdlock(9F), rwlock_tryrdlock(9F), rwlock_tryupgrade(9F), rwlock_trywrlock(9F),
rwlock_unlock(9F), rwlock_upgrade(9F), rwlock_wrlock(9F)
rwspin_alloc(9F)

NAME

rwspin_alloc - Dynamically allocate and initialize a reader-writer spinlock

SYNOPSIS

#include <sys/rwspin.h>
rwspin_t rwspin_alloc(
    char *name,
    rwspin_attr_t *attr,
    rwspin_attr_flg_t alloc_flag,
    unsigned int major_order,
    uintptr_t minor_order
);

PARAMETERS

name The reader-writer spinlock name.
attr Pointer to a reader-writer spinlock attribute (rwspin_attr_t) structure or NULL.
alloc_flag RWSPIN_WAITOK or RWSPIN_NOWAIT; these are mutually exclusive.
major_order The major order number that is used for deadlock detection in a debug kernel.
minor_order The order for locks with the same major order number. Frequently, the address of the structure being protected.

DESCRIPTION

The rwspin_alloc kernel function dynamically allocates and initializes a reader-writer spinlock. Reader-writer spinlocks support the following attributes:

RWSPIN_WAITOK
RWSPIN_NOWAIT

RETURN VALUES

<>NULL Pointer to the allocated spinlock.
NULL The allocation failed because of a lack of memory (in the case of RWSPIN_NOWAIT).

CONSTRAINTS

Must be called in a thread context and with no spinlocks held, if RWSPIN_WAITOK is specified.

EXAMPLE

#include <sys/rwspin.h>
rwspin_t *my_rwlock_p;

my_rwlock_p = rwspin_alloc ("My RWspin Lock", NULL, RWSPIN_WAITOK,
    MY_SPIN_MAJ_ORDER, NULL);
if (my_rwlock_p == NULL) {
    /*
     * Error management code
     */
}
/*
 * Code using the lock
 */
/* 
* Done with the lock 
*/
rwspin_dealloc (my_rwlock_p);

SEE ALSO

rwspin_attr_init(9F), rwspin_dealloc(9F), rwspin_owned(9F), rwspin_rdlock(9F), rwlock_rdunlock(9F),
rwspin_wrlock(9F), rwspin_wrunlock(9F)
NAME
rwspin_attr_init - Initialize optional reader-writer spinlock attributes

SYNOPSIS
#include <sys/rwspin.h>
void rwspin_attr_init(
    rwlock_attr_t *attr,
    size_t size
);

PARAMETERS
attr  A pointer to the reader-writer spinlock attribute (rwspin_attr_t) structure.
size  The size of the reader-writer spinlock attribute structure.

DESCRIPTION
The rwspin_attr_init routine initializes the reader-writer spinlock attribute structure. Kernel
modules can pass optional attributes to the reader-writer creation routines through an attribute
data structure. Before using this structure, kernel modules must initialize the structure to its
default values. After the data structure is initialized, the module can set the specific attributes
for the reader-writer spinlock and then call the creation routine.

This routine is provided for binary compatibility. If new attributes are added to the structure in
the future, you will not have to recompile your code. There are no attributes currently defined
for reader-writer spinlocks.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
#include <sys/rwspin.h>

rwspin_t *my_rwlock_p;
rwspin_attr_t my_attr;
/*
 * Set default values.
 */
rwspin_attr_init(&my_attr, sizeof(rwspin_attr_t));

my_rwlock_p = rwspin_alloc("My rwspin Lock", &my_attr, RWSPIN_WAITOK,
    MY_SPIN_MAJ_ORDER, NULL);

SEE ALSO
rwspin_alloc(9F), rwspin_dealloc(9F), rwspin_owned(9F), rwspin_rdlock(9F), rwlock_rdunlock(9F),
rwspin_wrlock(9F), rwspin_wrunlock(9F)
rwspin_dealloc(9F)

NAME

rwspin_dealloc - Destroy and deallocate a reader-writer spinlock

SYNOPSIS

#include <sys/rwspin.h>

void rwspin_dealloc(
    rwspin_t *rwslp
);

PARAMETERS

rwslp Pointer to the reader-writer spinlock.

DESCRIPTION

The rwspin_dealloc kernel function destroys and deallocates a reader-writer spinlock.

RETURN VALUES

None

CONSTRAINTS

Must be called with the lock unlocked.

EXAMPLE

#include <sys/rwspin.h>

rwspin_t *my_rwspin_p;

my_rwspin_p = rwspin_alloc("My RWspin Lock", NULL, RWSPIN_NOWAIT,
    MY_SPIN_MAJ_ORDER, NULL);
if (my_spin_p == NULL) {
    /* Error management code
        */
}
/*
 * Code using the lock
 */
/*
 * Done with the lock
 */
rwspin_dealloc (my_rwspin_p);

SEE ALSO

rwspin_alloc(9F), rwspin_attr_init(9F), rwspin_owned(9F), rwspin_rdlock(9F), rwlock_rdunlock(9F),
rwspin_wrlock(9F), rwspin_wrunlock(9F)
rwspin_owned(9F)

NAME

rwspin_owned - Determine the current lock mode of the lock.

SYNOPSIS

#include <sys/rwspin.h>

rwspin_owned_t rwspin_owned(
    rwspin_t *rwslp
);

PARAMETERS

rwslp Pointer to the reader-writer spinlock.

DESCRIPTION

The rwspin_owned kernel function determines the current lock mode for the calling processor.

RETURN VALUES

RWS_UNLOCKED The processor does not have the lock locked.
RWS_READLOCKED The processor has read access.
RWS_WRITELOCKED The processor has write access.

CONSTRAINTS

None

EXAMPLE

#include <sys/rwspin.h>

rwspin_t *lock_p;
header_t header;

data_t *add_to_list(data_t *data
{
    /*
     * Add to head of single linked list. The list is protected by lock_p.
     * Caller better have write access.
     */
    VASSERT (rwspin_owned(lock_p) == RWS_WRITELOCKED);

data->next = header.head_of_list;
header.head_of_list = data;
    return(OK);
}

SEE ALSO

rwspin_alloc(9F), rwspin_attr_init(9F), rwspin_dealloc(9F), rwspin_rdlock(9F), rwlock_rduplex(9F), rwspin_wrlock(9F), rwspin_rwunlock(9F)
**NAME**

rwspin_rdlock - Acquire a reader-writer spinlock for read access

**SYNOPSIS**

```c
#include <sys/rwspin.h>

void rwspin_rdlock(
    rwspin_t *rwslp
);
```

**PARAMETERS**

*rwslp* Pointer to the reader-writer spinlock.

**DESCRIPTION**

The `rwspin_rdlock` kernel function acquires a reader-writer spinlock for read access. If necessary, the processor busy-wait until the spinlock is available for reading. The processor must not already own the spinlock when you call this routine.

**RETURN VALUES**

None

**CONSTRAINTS**

None

**EXAMPLE**

```c
#include <sys/rwspin.h>

rwspin_t *lock_p;
header_t header;

data_t *search_list(int val)
{
    data_t *next;
    /*
     * Search the list for val. The list is protected by lock_p.
     */
    rwspin_rdlock(lock_p);
    next = header.head_of_list;
    while ((next!=NULL && (next->val != val))
        next = next->next;
    rwspin_rmunlock(lock_p);
    return(next);
}
```

**SEE ALSO**

rwspin_alloc(9F), rwspin_attr_init(9F), rwspin_dealloc(9F), rwspin_owned(9F), rwlock_rdunlock(9F), rwspin_wrlock(9F), rwspin_wrunlock(9F)
rwspin_rdunlock(9F)

NAME

rwspin_rdunlock - Release read access of a reader-writer spinlock

SYNOPSIS

#include <sys/rwspin.h>
void rwspin_rdunlock(
  rwspin_t *rwslp
);

PARAMETERS

rwslp Pointer to the reader-writer spinlock.

DESCRIPTION

The rwspin_rdunlock kernel function releases read access of a reader-writer spinlock. The processor must have the lock locked for read access.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

#include <sys/rwspin.h>

rwspin_t *lock_p;
header_t header;

data_t *search_list(int val)
{
  data_t *next;
  /*
   *Search the list for val. The list is protected by lock_p.
   */
  rwspin_rdlock(lock_p);
  next = header.head_of_list;
  while ((next!=NULL && (next->val != val))
    next = next->next;
  rwspin_rdunlock(lock_p);

  return(next);
}

SEE ALSO

rwspin_alloc(9F), rwspin_attr_init(9F), rwspin_dealloc(9F), rwspin_owned(9F), rwlock_rdlock(9F),
rwspin_wrlock(9F), rwspin_wrunlock(9F)
NAME

rwspin_wrlock - Acquire a reader-writer spinlock for write access

SYNOPSIS

#include <sys/rwspin.h>

void rwspin_wrlock(
    rwspin_t *rwslp
);

PARAMETERS

rwslp Pointer to the reader-writer spinlock.

DESCRIPTION

The rwspin_wrlock kernel function acquires a reader-writer spinlock for write access. If necessary, the processor busy-wait until the spinlock is available for writing. The processor must not already own the spinlock when you call this routine.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

#include <sys/rwspin.h>

rwspin_t *lock_p;
header_t header;

data_t *add_to_list(data_t *data
{
    /*
     * Add to head of single linked list. The list is protected by lock_p.
     */
    rwspin_wrlock(lock_p);
    data->next = header.head_of_list;
    header.head_of_list = data;
    rwspin_wrunlock(lock_p);

    return(OK);
}

SEE ALSO

rwspin_alloc(9F), rwspin_attr_init(9F), rwspin_dealloc(9F), rwspin_owned(9F), rwspin_rdlock(9F),
rwlock_rdunlock(9F), rwspin_wrunlock(9F)
rwspin_wrunlock(9F)

NAME

rwspin_wrunlock – Release a reader-writer spinlock for write access

SYNOPSIS

#include <sys/rwspin.h>

void rwspin_wrunlock(rwspin_t *rwslp);

PARAMETERS

rwslp Pointer to the reader-writer spinlock.

DESCRIPTION

The rwspin_wrunlock kernel function releases a reader-writer spinlock for write access.
The processor must own the spinlock for write access.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

#include <sys/rwspin.h>

rwspin_t *lock_p;
header_t header;

data_t *add_to_list(data_t *data
{
    /*
     * Add to head of single linked list. The list is protected by lock_p.
     */
    rwspin_wrlock(lock_p);
    data->next = header.head_of_list;
    header.head_of_list = data;
    rwspin_wrunlock(lock_p);

    return(OK);
}

SEE ALSO

rwspin_alloc(9F), rwspin_attr_init(9F), rwspin_dealloc(9F), rwspin_owned(9F), rwspin_rdlock(9F),
rwlock_rdunlock(9F), rwspin_wrunlock(9F)
seltrue(9F)

NAME

seltrue - Select driver entry point that returns true for traditional select flags.

SYNOPSIS

#include <sys/select.h>

int seltrue(
    dev_t dev,
    int flag
);

PARAMETERS

dev   Device number.
flag  Device select flags.

DESCRIPTION

The seltrue kernel function may be used as the driver entry point for the select system call. It returns true for the traditional select flag values of FREAD, FWRITE, and 0.

RETURN VALUES

The seltrue function returns the following values:
0 Device select flag is not a traditional select option.
-1 Device select flag is either FREAD, FWRITE, or 0.

CONSTRAINTS

None

EXAMPLES

drv_ops_t lpr0_drv_ops = { /* driver entry points */
    lpr0_open,          /* open */
    lpr0_close,         /* close */
    NULL,               /* strategy */
    NULL,               /* dump */
    NULL,               /* psize */
    NULL,               /* reserved */
    lpr0_read,          /* read */
    lpr0_write,         /* write */
    lpr0_ioctl,         /* ioctl */
    seltrue,            /* select */
    NULL,               /* option1 */
    0,                  /* pfilter */
    0,                  /* d_psize1 */
    0,                  /* d_drv_cb */
    NULL,               /* reserved entry point */
    0,                  /* device flags */
};

SEE ALSO

select(2)
NAME

selwakeup - Wake up a kernel thread sleeping on a select condition.

SYNOPSIS

#include <sys/select.h>

void selwakeup(
    struct kthread *threadp,
    int collision
);

PARAMETERS

threadp Pointer to the kernel thread to be awakened.
collision Indicates another kernel thread is sleeping on a select condition.

DESCRIPTION

The selwakeup kernel function wakes up a kernel thread sleeping on a select condition. If collision is zero, only the kernel thread pointed to by threadp is awakened. If collision is non-zero, all kernel threads sleeping on a select condition are awakened.

RETURN VALUES

None

CONSTRAINTS

Do not call while holding a spinlock of order >= SCHED_LOCK_ORDER.

EXAMPLE

#include <sys/select.h>
#include <sys/kthread.h>
#define MYSEL_COLL 1    /* my select collision flag */

static int
mydev_select(dev_t dev, int rw)
{
    struct kthread * kthreadp;
    ...*/
    /* Check if another kernel thread is already
     * sleeping on a select condition with the driver.
     */
    kthreadp = myselstruct->thread;
    if (kthreadp &
        waiting_in_select (kthreadp) { /*
          * There is going to be a collision of multiple
          * threads sleeping on a select condition.
          */
          myselstruct->selflag |= MYSEL_COLLISION;
    } else { /*
          * Save the kernel thread pointer of this
          * thread that will sleep on a select condition.
          */
          myselstruct->thread = kthreadp_self();
    }
    ...
static void mydev_selwakeup(void)
{
    selwakeup(myselstruct->thread,
              myselstruct->selflag & MYSEL_COLLISION);
    myselstruct->thread = NULL;
    myselstruct->selflag &= ~MYSEL_COLLISION;
}

SEE ALSO

waiting_in_select(9F)
NAME

spin_alloc - Allocate and initialize a spinlock

SYNOPSIS

#include <sys/spin.h>

spin_t spin_alloc(
    char *name,
    spin_attr_t *attr,
    spin_alloc_flg_t alloc_flags,
    unsigned int major,
    uintptr_t minor
);

PARAMETERS

name The spinlock name.
attr If not NULL, a pointer to the spinlock attribute (spin_attr_t) structure. Only NULL is supported.
alloc_flags Either SPIN_WAITOK or SPIN_NOWAIT. They are mutually exclusive.
major The major order number used for deadlock detection. Use this only in a debug kernel.
minor The order for locks with the same major order number. Frequently, the address of the structure that the lock is protecting. Use this only in a debug kernel.

DESCRIPTION

The spin_alloc routine dynamically allocates and initializes a spinlock. It returns a pointer to the spinlock.

The attr parameter specifies the optional attributes to be assigned to the lock. If the attr parameter is NULL, the default attributes are assigned to the lock. Currently, there are no optional attributes defined for spinlocks. The major and minor numbers are the orders used for deadlock detection.

RETURN VALUES

non-NULL A pointer to the allocated spinlock.
NULL The allocation failed because of a lack of memory (if SPIN_NOWAIT is specified).

CONSTRAINTS

You must call this routine only from a thread context and hold no spinlocks, if SPIN_WAITOK is specified.

EXAMPLE

#include <sys/spin.h>

spin_t *my_spin_p;

my_spin_p = spin_alloc("my_spin_lock", NULL, SPIN_NOWAIT,
    MY_SPIN_MA Fresno0y, NULL);

if (my_spin_p == NULL {
    /* Error management code.
    */
spin_dealloc(my_spin_p);
    /* Done with the spinlock, */

SEE ALSO

spin_attr_init(9F), spin_dealloc(9F), spin_lock(9F), spin_unlock(9F), spin_trylock(9F), spin_owned(9F), spin_locks_held(9F)
NAME

spin_attr_init - Initialize optional spinlock attributes

SYNOPSIS

#include <sys/spin.h>

void spin_attr_init(
    spin_attr_t *attr,
    size_t size
);

PARAMETERS

attr A pointer to the spinlock attribute (spin_attr_t) structure. No attributes are currently defined.

size The size of the spinlock attribute structure.

DESCRIPTION

The spin_attr_init routine initializes the spinlock attribute structure. Kernel modules can pass optional attributes to the spinlock creation routines through an attribute data structure. Before using this structure, kernel modules must initialize the structure to its default values. After the structure is initialized, the module can set the specific attributes for the spinlock and call the creation routine.

This routine is provided for binary compatibility. If new attributes are added to the attribute structure in the future, you will not have to recompile your code. Currently, there are no optional attributes defined for spinlocks.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

#include <sys/spin.h>

spin_t *lock_p;
spin_attr_t my_attr;

/* No attributes are currently defined. Prepare for that event. */
spin_attr_init(&my_attr, sizeof(spin_attr_t));

/* Set default values. */
lock_p = spin_alloc("My Spin Lock", &my_attr, SPIN_WAITOK,
    MY_MAJOR_NUMBER, NULL);

SEE ALSO

spin_alloc(9F), spin_dealloc(9F), spin_lock(9F), spin_unlock(9F), spin_trylock(9F), spin_owned(9F), spin_locks_held(9F)
NAME
spin_dealloc - Destroy and deallocate a spinlock

SYNOPSIS
#include <sys/spin.h>
void spin_dealloc(
    spin_t *sp
);

PARAMETERS
sp  A pointer to the spinlock to be deallocated.

DESCRIPTION
The spin_dealloc routine destroys and deallocates a spinlock.

RETURN VALUES
None

CONSTRAINTS
You must call this routine only with the spinlock unlocked.

EXAMPLE
#include <sys/spin.h>
spin_t *my_spin_p;

my_spin_p = spin_alloc("my_spin_lock", NULL, SPIN_NOWAIT,
    MY_SPIN_MAJ_ORDER, NULL);

if (my_spin_p == NULL {
    /* Error management code.
    */
    spin_dealloc(my_spin_p);
    /* Done with the spinlock, */

SEE ALSO
spin_alloc(9F), spin_attr_init(9F), spin_lock(9F), spin_unlock(9F), spin_trylock(9F), spin_owned(9F),
spin_locks_held(9F)
spin_lock(9F)

NAME

spin_lock -- Acquire (lock) a spinlock.

SYNOPSIS

#include <sys/spin.h>

void spin_lock(
    spin_t *sp
);

PARAMETERS

sp Pointer to the spinlock to be locked.

DESCRIPTION

The spin_lock kernel function attempts to acquire (lock) the spinlock pointed to by sp. If
necessary, spin_lock spins with interrupts masked (SPLSYS) until the lock is available.
Interrupts are masked upon return.

Spinlocks are the basic locking primitive used by the kernel for short-term locks. When a thread
acquires a spinlock, the thread’s current processor becomes the effective owner until the spinlock
is released. Threads (processors) waiting to acquire an owned spinlock will spin while waiting;
they do not block. For the duration that a processor owns a spinlock, external interrupts to the
processor are disabled.

RETURN VALUES

None

CONSTRAINTS

The processor must not own the spinlock.
Do not call spin_lock at an spl level higher than SPLSYS (for example, SPL7).

EXAMPLE

#include <sys/spin.h>

spin_t *lock_p;
header_t header;

int add_to_list(int flags, data_t *data)
{
    /*
    * Acquire the spinlock for my driver.
    */
    spin_lock(lock_p);
    /* Add to head of single linked list.
    * The list is protected by lock_p.
    */
    data->next = header.head_of_list;
    header.head_of_list = data;
    /*
    * Release the spinlock when done.
    */
    spin_unlock(lock_p);
    return (OK);
}
SEE ALSO

spin_alloc(9F), spin_attr_init(9F), spin_dealloc(9F), spin_unlock(9F), spin_trylock(9F), spin_owned(9F), spin_locks_held(9F)
NAME

spin_locks_held - Determine whether the processor owns any type of spinlock

SYNOPSIS

#include <sys/spin.h>

int spin_locks_held(void);

PARAMETERS

None

DESCRIPTION

The spin_locks_held kernel function determines whether the current processor owns any type of spinlock.

RETURN VALUES

<0 The processor has locked a spinlock or reader-writer spinlock.
0 The processor does not own any type of spinlock.

CONSTRAINTS

None

EXAMPLE

#include <sys/spin.h>

int can_block;

can_block = !spin_locks_held();
/*
 * Cannot block if we own any spinlocks.
 */

SEE ALSO

spin_alloc(9F), spin_attr_init(9F), spin_dealloc(9F), spin_unlock(9F), spin_trylock(9F), spin_owned(9F), spin_locks_held(9F)
NAME
spin_owned – Test if processor owns a spinlock.

SYNOPSIS
#include <sys/spin.h>
int spin_owned(
spin_t *sp
);

PARAMETERS

sp Pointer to the spinlock to be tested.

DESCRIPTION
The spin_owned kernel function tests whether the spinlock pointed to by sp is owned by the current processor.

RETURN VALUES
<>0 The processor owns the spinlock.
0 The processor does not own the spinlock.

CONSTRAINTS
None

EXAMPLE
#include <sys/spin.h>

spin_t *lock_p;
header_t header;

int add_to_list(int flags, data_t *data)
{
    /* Better own the spinlock because
     * we are going to update the list.
     */
    VASSERT (spin_owned(lock_p));

    /* Add to head of single linked list.
     * The list is protected by lock_p.
     */
    data->next = header.head_of_list;
    header.head_of_list = data;

    return (OK);
}

SEE ALSO
spin_alloc(9F), spin_attr_init(9F), spin_dealloc(9F), spin_lock(9F), spin_unlock(9F), spin_trylock(9F), spin_locks_held(9F)
spin_trylock(9F)

NAME
spin_trylock - Conditionally acquire (lock) a spinlock.

SYNOPSIS
#include <sys/spin.h>
int spin_trylock(
    spin_t *sp
);

PARAMETERS
sp Pointer to the spinlock to be locked.

DESCRIPTION
The spin_trylock kernel function conditionally attempts to acquire (lock) the spinlock pointed to by sp. If the lock is not available, the function fails.

RETURN VALUES
<0 The processor successfully locked (acquired) the spinlock.
0 The lock failed.

CONSTRAINTS
None

EXAMPLE
#include <sys/spin.h>

spin_t *lock_p;
header_t header;

int add_to_list(int flags, data_t *data)
{
    if (flags == NO_SPIN) {
        /*
         * Cannot wait for the lock.
         */
        if (!spin_trylock(lock_p) {
            return (FAILED_TO_DO_WORK);
        }
    }
    else spin_lock(lock_p);
    /* Add to head of single linked list.
     * The list is protected by lock_p.
     */
    data->next = header.head_of_list;
    header.head_of_list = data;
    /*
     * Release the spinlock when done.
     */
    spin_unlock(lock_p);
    return (OK);
}

SEE ALSO
spin_alloc(9F), spin_attr_init(9F), spin_dealloc(9F), spin_lock(9F), spin_unlock(9F), spin_owned(9F), spin_locks_held(9F)
NAME

spin_unlock - Unlock a spinlock.

SYNOPSIS

#include <sys/spin.h>

void spin_unlock(
    spin_t *sp
);

PARAMETERS

sp  Pointer to the spinlock to be unlocked.

DESCRIPTION

The spin_unlock kernel function unlocks the spinlock pointed to by sp and grants ownership to a waiting processor. Only the processor that owns the spinlock can unlock it.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

#include <sys/spin.h>

spin_t *lock_p;
header_t header;

int add_to_list(int flags, data_t *data)
{
    /*
     * Acquire the spinlock for my driver.
     */
    spin_lock(lock_p);
    /* Add to head of single linked list.
     * The list is protected by lock_p.
     */
    data->next = header.head_of_list;
    header.head_of_list = data;
    /*
     * Release the spinlock when done.
     */
    spin_unlock(lock_p);
    return (OK);
}

SEE ALSO

spin_alloc(9F), spin_attr_init(9F), spin_dealloc(9F), spin_lock(9F), spin_trylock(9F), spin_owned(9F), spin_locks_held(9F)
NAME

snprintf - Kernel version of printf.

SYNOPSIS

#include <sys/kern_svcs.h>

int snprintf(
    char *str,
    int len,
    const char *fmt,
    ...
);

PARAMETERS

str Address of buffer to hold the formatted string.
len Specifies the length of the formatted string, including the trailing '\0'.
fmt A set of printing characters and limited conversion specifications, as defined in snprintf(3S).
... An argument corresponding to a format conversion specification, as defined in snprintf(3S).

DESCRIPTION

The snprintf kernel function is similar to the C library snprintf routine (see snprintf(3S)).
The snprintf writes information to the str array.
The kernel snprintf routine can accept the following formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Printed Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>%!</td>
<td>%; no argument</td>
</tr>
<tr>
<td>%c</td>
<td>Character from integer argument</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal from integer argument</td>
</tr>
<tr>
<td>%lx</td>
<td>The argument x is long integer; x is one of x,d,u, or o.</td>
</tr>
<tr>
<td>%o</td>
<td>Unsigned octal from integer argument</td>
</tr>
<tr>
<td>%s</td>
<td>Characters from string argument</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal from integer argument</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal from integer argument</td>
</tr>
</tbody>
</table>

These formats are the same as in snprintf(3S). Other formats specified in snprintf(3S) are not supported.

RETURN VALUES

The snprintf function returns the length of formatted string.

CONSTRAINTS

None

SEE ALSO

msg_printf(9F),printf(9F),snprintf(3S)
NAME

strcat - Concatenate two strings.

SYNOPSIS

#include <sys/kern_svcs.h>

char *strcat(
    char *buf,
    const char *str
);

PARAMETERS

buf  Pointer to a string.
str  Pointer to a string.

DESCRIPTION

The strcat kernel function appends string buf to string char.

RETURN VALUES

The strcat function returns buf.

CONSTRAINTS

None

SEE ALSO

strcpy(9F), strlen(9F), strncmp(9F), strncpy(9F)
**NAME**

strcmp - Compare two strings.

**SYNOPSIS**

```c
#include <sys/kern_svcs.h>

int strcmp(
    const char *s1,
    const char *s2 );
```

**PARAMETERS**

- **s1**  Pointer to a character string.
- **s2**  Pointer to a character string.

**DESCRIPTION**

The `strcmp` kernel function compares two null-terminated strings. If the strings are the same, zero is returned. If the strings are different, the integer value of `(*s1-*s2)` is returned.

**RETURN VALUES**

- **0**  The strings are identical.
- **<>0**  The strings are different.

**CONSTRAINTS**

None

**EXAMPLE**

```c
char *
scsi_decode_opcode(dev_type, op)
int dev_type;
int op;
{
    struct scsi_opcode *entry = scsi_opcode[op];
    int dev_type_bit = 1 << dev_type;
    int i;

    static int initialized;

    /*
    * Do a sanity check on the scsi_opcode table.
    */
    if (!initialized)
    {
        i = strcmp(scsi_opcode[CMDread_element_status][0].op_str,
                    "Read Element Status");
        if (i == 0)
            initialized = 1;
        else
            return "BadTable";
    }

    .
    .
    .
}
```
SEE ALSO

strlen(9F), strncmp(9F), strncpy(9F)
NAME

strcpy - Copy the characters from one string to another string.

SYNOPSIS

```
#include <sys/kern_svcs.h>

char *strcpy(
    char *s1,
    const char *s2
);
```

PARAMETERS

- **s1** Pointer to the destination string.
- **s2** Pointer to the source string.

DESCRIPTION

The `strcpy` kernel function copies the characters from the string pointed to by `s2` to the string pointed to by `s1`, terminating at the first null character in `s2`. A pointer to the string `s1` is returned. No bounds checking is done.

RETURN VALUES

The `strcpy` function returns the string pointed to by `s1`.

CONSTRAINTS

None

EXAMPLE

```
{
    .
    .
    if ( slot_is_isa )
        strcpy(&desc[0],"ISA card ");
    else {
        strcpy(desc,"EISA card ");
        cvt_eisa_id_to_ascii( valid_card_id, id_str);
        /* id_str returns 8 bytes with last byte null */
        strcpy(&desc[strlen(desc)], id_str);
    }
    .
    .
}
```

SEE ALSO

`strcmp(9F), strlen(9F), strncmp(9F), strncpy(9F)`
NAME

strlen - Gets the number of nonnull bytes in a string.

SYNOPSIS

#include <sys/kern_svcs.h>

int strlen(
    char *s
);

PARAMETERS

s    Pointer to the string.

DESCRIPTION

The strlen kernel function returns the number of bytes in the string, up to but not including
the first null.

RETURN VALUES

The strlen function returns the number of non-null bytes in the string.

CONSTRAINTS

None

EXAMPLES

/ *
   * Hex dump a chunk of data with header into the dmesg
   * buffer.
   */

msg_dump(label, buf, n)
    char *label;
    u_char *buf;
    int n;
{
    int i, j, indent;

    msg_printf("\t%s:", label);
    indent = strlen(label) + 1;

    for (i = 0; i < n; i++)
    {
        if ((i & 0xf) == 0 & i != 0)
        {
            msg_printf("\n\t");
            for (j = 0; j < indent; j++)
                msg_printf(" ");
        }
        msg_printf(" %02x", buf[i]);
    }
    msg_printf("\n");
}

SEE ALSO

strcmp(9F), strcpy(9F), strncmp(9F), strncpy(9F)
**NAME**

strncmp - Compare the first n characters of two strings.

**SYNOPSIS**

```c
#include <sys/kern_svcs.h>

int strncmp(
    char *s1,
    char *s2,
    int n
);
```

**PARAMETERS**

- `s1` Pointer to a string.
- `s2` Pointer to a string.
- `n` Number of bytes to compare.

**DESCRIPTION**

The `strncmp` kernel function compares the first `n` characters of two null-terminated strings. If the strings are the same, zero is returned. If the strings are different, the integer value of `(*s1-*s2)` is returned.

**RETURN VALUES**

- `0` The strings are equivalent.
- `<0` The strings are different.

**CONSTRAINTS**

None

**EXAMPLE**

```c
{
    
    if (strncmp(driver_name, "enet",4) !=0)
        /* The driver is not ENET */
        return -1;
    else {
        
        }
    
    }
```

**SEE ALSO**

`strcmp(9F), strlen(9F), strncmp(9F), strncpy(9F)`
NAME
strncpy - Copy characters between strings.

SYNOPSIS
#include <sys/kern_svcs.h>

char *strncpy
  (char *s1,
   char *s2,
   int n
  );

PARAMETERS
s1 Pointer to a string.
s2 Pointer to a string.
n  Number of bytes to copy.

DESCRIPTION
The strncpy kernel function copies the characters from the string pointed to by s2 to the string
pointed to by s1. Copying terminates at n bytes or after a NULL character which ever occurs
first. No bounds checking is done.

RETURN VALUES
A pointer to the string s1 is returned. The strncpy function returns a pointer to the copied (s1)
string.

CONSTRAINTS
None

EXAMPLES
{
  .
  .
  /*
   * Get the first four bytes of the inquiry data
   * for dev_id and the vendor and product id for desc
   */
  strncpy(desc, iqr_data->inq2.vendor_id, 8);
  strncpy(&desc[8], iqr_data->inq2.product_id, 16);
  desc[24] = '\0';
  .
  .
}

SEE ALSO
string(3C), strcmp(9F), strlen(9F), strncmp(9F), strcpy(9F)
NAME

timeout - Execute a callout function after a specified length of time.

SYNOPSIS

#include <sys/param.h>
#include <sys/callout.h>
callout_t *timeout(
    int (*func)(),
    caddr_t arg,
    int t
);

PARAMETERS

  func  Function to call when the time value t expires.
  arg   Argument passed to the callout function func.
  t     Time value in number of clock ticks.

DESCRIPTION

The timeout kernel function executes the specified callout function func after t clock ticks have expired. Execution of func takes place in an interrupt context at priority level two where external interrupts to the processor are enabled.

The func parameter is a pointer to a function that takes one argument. Although the prototype declares the function to return an integer value, the kernel does not make use of the return value. The arg parameter is passed as the one argument to func.

The t parameter specifies the number of clock ticks to wait before calling func. To express time in seconds, multiply t by Hz, where Hz is defined as the number of clock ticks per second in <sys/param.h>.

The call to timeout returns immediately without waiting for the time value to expire. The timeout can be canceled by making a corresponding call to untimeout.

RETURN VALUES

The timeout function returns a pointer to a callout structure. The caller must treat it as an opaque handle, and not try to change the contents of the structure.

CONSTRAINTS

Do not call while holding a spinlock of order >= CALLOUT_LOCK_ORDER.

EXAMPLES

/*
 * Set a timeout to call my_timeout_func passing my_arg
 * after 5 seconds have expired.
 */
(void)timeout(my_timeout_func, my_arg, 5*HZ);

static int
my_timeout_func(caddr_t arg)
{
    ...
WARNINGS

Each call to \texttt{timeout} allocates a callout resource, and the resource is not released until the time value expires or the timeout is canceled. The kernel may panic if it has no more callout resources available, and is unable to dynamically allocate more.

SEE ALSO

\texttt{untimeout(9F)}, \texttt{delay(9F)}. 
**NAME**

uio - Data descriptor for scatter/gather I/O requests.

**SYNOPSIS**

```
#include <sys/uio.h>
```

**DESCRIPTION**

The `uio` structure specifies an I/O request that can be fragmented into multiple buffers (scatter/gather I/O). The `uio` structure contains a pointer to an array of `iovec` structures, each specifying the base address and length of a buffer. Buffers may be in either user space or kernel space.

For `read` and `write` system calls, the kernel allocates and fills out a `uio` structure pointing to a single `iovec` structure. The `uio` structure for `readv` and `writev` system calls points to an array of `iovec` structures. The `uio` structure is passed to the `driver_read` or `driver_write` entry point of the target driver.

Drivers must not access fields in the `uio` structure directly. The `physio` kernel function is called to perform raw, unbuffered I/O and `uiomove` is called to copy the specified buffer(s) to/from a kernel buffer. Fields in the `uio` structure are updated by `physio` and `uiomove`.

**STRUCTURE MEMBERS**

The `uio` structure is defined in `<sys/uio.h>`, and has the following fields:

**Table 1-3 uio Structure Fields**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uio_iov</code></td>
<td><code>struct iovec *</code></td>
</tr>
<tr>
<td><code>uio_iovcnt</code></td>
<td><code>size_t</code></td>
</tr>
<tr>
<td><code>uio_seg</code></td>
<td><code>uint32_t</code></td>
</tr>
<tr>
<td><code>uio_resid</code></td>
<td><code>long</code></td>
</tr>
<tr>
<td><code>uio_fpflags</code></td>
<td><code>uint32_t</code></td>
</tr>
<tr>
<td><code>uio_offset</code></td>
<td><code>off_t</code></td>
</tr>
</tbody>
</table>

- **`uio_iov`**: Pointer to an array of `iovec` structures. Each `iovec` structure contains the base address and length of a buffer.
- **`uio_iovcnt`**: Number of entries in the array of `iovec` structures.
- **`uio_seg`**: Indicates the addresses of the buffers are in user space (UIOSEG_USER) or kernel space (UIOSEG_KERNEL).
- **`uio_resid`**: Number of bytes remaining to be transferred. Initially, this equals the sum of all the buffer lengths.
- **`uio_fpflags`**: Flags that indicate whether the driver must not wait:
  - **FNDELAY**: The driver must not wait if the requested data transfer cannot be done immediately. The request must be terminated without returning an error code.
  - **FNBLOCK**: The driver must not wait if the requested data transfer cannot be done immediately. The request must be terminated and return EAGAIN as the error code.
**uio_offset** Starting logical byte address on the device where the data transfer is to occur. Applicability of this field to the driver is device dependent. The *uio_offset* is usually applicable only to devices capable of seeking.

**SEE ALSO**

`iovec(9S), open(2), physio(9F), read(2), readv(2), uiomove(9F), write(2), writev(2)`
uiomove(9F)

NAME

uiomove - Copy data between memory in kernel space and the space specified by a uiop structure.

SYNOPSIS

#include <sys/uio.h>

int uiomove(
    caddr_t addr,
    int n,
    int flag,
    struct uio *uiop
);

PARAMETERS

addr Address of kernel memory.
n Number of bytes to copy.
flag Copy direction: UIO_READ or UIO_WRITE.
uiop Pointer to a uio structure.

DESCRIPTION

The uiomove kernel function copies data between memory in kernel space and the space specified by the uiop structure pointed to by uiop. The uiomove function is typically used to copy data between user space and kernel space.

The addr parameter is the address of kernel memory to/from which data are to be copied. Kernel memory can be allocated by calling kmalloc or geteblk. Do not allocate kernel memory on the stack if the allocation size is greater than 128 bytes, as this can potentially lead to a stack overflow.

The n parameter is the number of bytes to copy. Typically, n is set to the value in uiop->uio_resid. If this value is greater than the size of allocated kernel memory, multiple calls to uiomove with a smaller value of n will be required.

The flag parameter determines the direction of the copy: UIO_READ copies data from addr, UIO_WRITE copies data to addr.

The uiop parameter is a pointer to the uiop structure. If the uio_segflag field in the uiop structure is UIO_USERSPACE, user space is to be accessed and uiomove must be called in the user context. If uio_seg is UIO_SYSSPACE, kernel space is to be accessed and uiomove can be called in any user context or in an interrupt context.

The uio_iov field in the uiop structure points to an iovec structure. If uio_iovcnt is greater than 1, uio_iov points to an array of iovec structures. The uiomove function walks through the array of iovec structures as it performs the copy.

If the copy is successfully completed, fields in the uiop structure are updated as follows:

- uio_offset is increased by n.
- uio_resid is decreased by n.

For a single iovec structure (uio_iovcnt = 1), a successful completion will update the iovec as follows:

- uio_iov->iov_base is increased by n.
- uio_iov->iov_len is decreased by n.

For an array of iovec structures (uio_iovcnt > 1), a successful completion will increment each iov_base field appropriately and decrement each iov_len field appropriately.
RETURN VALUES
0Successful completion.
<>0Error.

CONSTRAINTS
If the \texttt{uio\_segflag} field in the \texttt{uio} structure is \texttt{UIO\_USERSPACE}, user space is to be accessed and \texttt{uiomove} must be called in the user context. If \texttt{uio\_seg} is \texttt{UIO\_SYSSPACE}, kernel space is to be accessed and \texttt{uiomove} can be called in any user context or in an interrupt context.

EXAMPLE

\begin{verbatim}
static int
mydriver_write(dev_t dev, struct uio * uiop)
{
    struct buf * bp;
    int count = uiop->uio iov iov len;

    bp = geteblk(count);
    (void)uiomove(bp->b un b_addr, count, UIO_WRITE, uio);
    brelse(bp);
    ...}
\end{verbatim}

SEE ALSO

\texttt{brelse(9F), geteblk(9F)}
unmap_mem_from_host(9F)

NAME

unmap_mem_from_host - Unmap physical bus from host virtual space.

SYNOPSIS

#include <sys/wsio.h>

int unmap_mem_from_host(
    struct isc_table_type *isc,
    caddr_t virt_addr,
    size_t size
);

PARAMETERS

isc             The ISC pointer used in the previous map_mem_to_host call.
virt_addr       The virtual address previously obtained with a map_mem_to_host call.
size            The size of the memory space in bytes.

DESCRIPTION

The map_mem_to_host kernel routine is used to unmap previously mapped virtual space. Hardware disables caching for all EISA data accesses, including EISA memory space.

RETURN VALUES

The map_mem_to_host function returns an indeterminate value. Treat it as if it was a void function.

CONSTRAINTS

None

SEE ALSO

map_mem_to_host(9F)
NAME
untimeout - Cancel a previous timeout request.

SYNOPSIS
#include <sys/callout.h>
int untimeout(
    int (*func)(),
    caddr_t arg
);

PARAMETERS
func The func parameter from the previous timeout request.
arg The arg parameter from the previous request.

DESCRIPTION
The untimeout kernel function cancels a previous timeout request made by a call to timeout. The func and arg parameters are compared against the list of timeout requests waiting to expire. If a match is found, the timeout request is canceled by taking it off the list and releasing allocated resources.

RETURN VALUES
>=0 Number of clock ticks remaining in the cancelled timeout request.
-1 Timeout request not found. This is a normal return if the timeout request has already expired or has been dispatched to be executed.

CONSTRAINTS
Do not call while holding a spinlock of order >= CALLOUT_LOCK_ORDER.

EXAMPLES
/ *
 * Cancel a previous timeout to call my_timeout_func()
 */
(void) untimeout(my_timeout_func, my_arg);

static int
my_timeout_func(caddr_t arg)
{
    ...
}

SEE ALSO
timeout(9F).
NAME
user_iomap - Maps I/O memory to a user address.

SYNOPSIS

```
#include <sys/iomap.h>

void *
user_iomap(
    void *vaddr,
    void *paddr,
    pgcnt_t count
);
```

PARAMETERS

- **vaddr**: Virtual address requested. must be NULL.
- **paddr**: Physical address of the I/O memory location from the view of the processor.
- **count**: Count of base pages to map.

DESCRIPTION

The `user_iomap` function memory maps into shared user address space a range of contiguous pages in I/O address space starting at the physical address `paddr` for `count` pages. The value returned is the user virtual address that maps onto the base of the I/O memory location specified by `paddr` and `count`.

The virtual address returned will be appropriate to the user process: a 32-bit address is returned for 32-bit processes and a 64-bit address is returned for 64-bit processes. The call to `user_iomap` must be made in the context of the user process; the call cannot be made in interrupt context.

The physical address is the 64-bit physical address from the view of the processor, and the address must be pagesize aligned. The pagesize can be obtained by calling `iomap_pagesize`.

The 64-bit physical address `paddr` is not necessarily the same address value that is obtained from the Base Address Register (BAR) of a PCI device. The caller may need to convert the BAR value to the appropriate 64-bit physical address value used by the processor.

You can call the `user_iomap` function multiple times for the same I/O memory range to allow access to the range by multiple user processes. For each call made to `user_iomap`, there must be a corresponding call made to `user_iounmap` in the context of each user process before the calling module unloads, or the module must wait for each user process to terminate, in which case the unmapping is done by the kernel.

It is normal for a call to `user_iomap` to fail. Possible reasons for failure include:

- The pages are mapped for kernel-only access with `PROT_KRW` and the user virtual address is equivalent to the kernel virtual address. If the pages are mapped by the WSIO driver environment or mapped using `wsio_map_reg`, protection is set to `PROT_KRW`.
- The pages overlap with an I/O memory range that has already been mapped by a previous call to `user_iomap` or `kernel_iomap`. To successfully map an I/O memory range multiple times, the mappings must cover exactly the same range (same `paddr` and `count` parameters).

A call to `user_iomap` might also fail if the driver previously called `wsio_map_reg` or if the WSIO driver environment has mapped the I/O memory on behalf of the driver. If the driver's `isc->if_reg_ptr` is not NULL, WSIO has mapped for kernel-only access (up to 8 KB) of the first PCI BAR in the device that specifies a memory range.

RETURN VALUES

- USER virtual address.
CONSTRANTS

Must be called in the user context.
Do not call in interrupt context.
Do not call while holding a spinlock.

EXAMPLES

    void  *physaddr;
    void  *useraddr;
    size_t pagesize;

    /*
    * Get the physical address as seen by the processor.
    * Verify that the physical address is pagesize aligned.
    */
    physaddr = xyz;
    pagesize = iomap_pagesize();
    VASSERT(((uintptr_t)physaddr % pagesize) == 0);

    /*
    * Memory map the user virtual address onto a page of the
    * I/O memory location.
    */
    useraddr = user_iomap(NULL, physaddr, 1L);

    if (useraddr == NULL) {
        /*
        * Failed to map user virtual address.
        * Add error handling code here.
        */
    }

SEE ALSO

iomap_pagesize(9F), kernel_iomap(9F), kernel_iounmap(9F), user_iomap_private(9F), user_iounmap(9F).
user_iomap_private(9F)

NAME

user_iomap_private - Maps I/O memory to a private user address.

SYNOPSIS

#include <sys/iomap.h>

void *user_iomap_private(
    void *vaddr,
    void *paddr,
    pgcnt_t count,
    uip_flags_t flags
);

PARAMETERS

vaddr Virtual address requested. must be NULL.
paddr Physical address of the I/O memory location from the view of the processor.
count Count of base pages to map.
flags Private mapping flags: UIP_NO_INHERIT and UIP_READ_ONLY.

DESCRIPTION

The user_iomap_private function memory maps into private user address space a range of contiguous pages in I/O address space starting at the physical address paddr for count pages. The value returned is the private user virtual address that maps onto the base of the I/O memory location specified by paddr and count.

The virtual address returned will be appropriate to the user process: a 32-bit address is returned for 32-bit processes and a 64-bit address is returned for 64-bit processes. The call to user_iomap_private must be made in the context of the user process; the call cannot be made in interrupt context.

Access to the I/O memory is typically inherited by a child process after the parent process calls fork. To disable the child from inheriting access, set the flags parameter to UIP_NO_INHERIT.

To limit user access rights to read-only, set the flags parameter to UIP_READ_ONLY. If this flag is not set, the user process is granted read-write access.

The UIP_DEFAULT flag specifies the default flags parameter settings, allowing inheritance and setting access rights to read-write.

The physical address is the 64-bit physical address from the view of the processor, and the address must be pagesize aligned. The pagesize can be obtained by calling iomap_pagesize.

The 64-bit physical address paddr is not necessarily the same address value that is obtained from the Base Address Register (BAR) of a PCI device. The caller may need to convert the BAR value to the appropriate 64-bit physical address value used by the processor.

You can call the user_iomap_private function multiple times for the same I/O memory range to allow access to the range by multiple user processes. For each call made to user_iomap_private, there must be a corresponding call made to user_iounmap in the context of each user process before the calling module unloads, or the module must wait for each user process to terminate, in which case the unmapping is done by the kernel.

It is normal for a call to user_iomap_private to fail. Possible reasons for failure include:
- The pages are mapped for kernel-only access with PROT_KRW.
- The pages overlap with an I/O memory range that has already been mapped by a previous call to user_iomap_private or kernel_iomap. To successfully map an I/O memory...
range multiple times, the mappings must cover exactly the same range (same paddr and count parameters).

RETURN VALUES

<>NULL User virtual address.
NULL Error. Mapping failed.

CONSTRAINTS

Must be called in the user context.
Do not call while holding a spinlock.

EXAMPLES

void  *physaddr;
void  *useraddr;
size_t pagesize;

/*
 * Get the physical address as seen by the processor.
 * Verify that the physical address is pagesize aligned.
 */
physaddr = xyz;
pagesize = iomap_pagesize();
VASSERT(((uintptr_t)physaddr % pagesize) == 0);

/*
 * Memory map the user virtual address onto a page of the
 * I/O memory location. The user virtual address will be
 * allocated from user private address space.
 */
useraddr = user_iomap)private(NULL, physaddr, 1L, UIP_DEFAULT);

if (useraddr == NULL) {
    /*
     * Failed to map user virtual address.
     * Add error handling code here.
     */
}

SEE ALSO

iomap_enable_wc(9F), iomap_pagesize(9F), kernel_iomap(9F), kernel_iounmap(9F), user_iomap(9F),
user_iounmap(9F).
user_iounmap(9F)

NAME

user_iounmap - Unmap I/O memory for the assigned user address.

SYNOPSIS

#include <sys/iomap.h>

int user_iounmap(
    void *vaddr,
    pgcnt_t count
);

PARAMETERS

vaddr Virtual address returned by user_iomap.
count Count of pages to unmap.

DESCRIPTION

The user_iounmap function removes the user virtual address assigned to map a range of I/O memory pages. If the same virtual address is assigned to multiple user processes, the address will remain valid for the other user processes.

The vaddr parameter is the virtual address returned by user_iomap, and the count parameters must match the count parameter passed to user_iomap.

RETURN VALUES

0 Success.
1 Failure.

CONSTRAINTS

Must be called in the user context.
Do not call in interrupt context.
Do not call while holding a spinlock.

EXAMPLES

void    *physaddr;
void    *useraddr;
pgcnt_t count;
int     retval;

...

  /*
   * Memory map the user virtual address onto the I/O memory location for count number of pages.
   */
  useraddr = user_iomap(NULL, physaddr, count);
  ...

  /*
   * Remove the mapping for the user virtual address.
   */
  retval = user_iounmap(useraddr, count);
  VASSERT(retval == 0);
SEE ALSO

iomap_enable_wc(9F), iomap_pagesize(9F), kernel_iomap(9F), kernel_iounmap(9F), user_iomap(9F), user_iomap_private(9F).
vm_phys_pagesize(9F)

NAME

vm_phys_pagesize -- Get the physical page size corresponding to a global virtual address (GVA).

SYNOPSIS

#include <sys/pas.h>

size_t vm_phys_pagesize(
    space_t space,
    void *vaddr
);

PARAMETERS

space   Space ID part of the GVA.
vaddr   Offset part of the GVA.

DESCRIPTION

The vm_phys_pagesize function returns the size (in bytes) of the page corresponding to the specified GVA. The GVA can be either user or kernel virtual address. If the returned value is zero, the space and vaddr inputs are not translated. The caller can use the page size returned to align the virtual address and determine the starting address for the page.

RETURN VALUES

0  GVA does not have a translation.
<>0  Size (in bytes) of the physical page.

CONSTRAINTS

None

EXAMPLES

void      *vaddr;
space_t    space;
size_t     size_in_bytes;

vaddr = get user buffer virtual address
space = ldusid(vaddr);
size_in_bytes = vm_phys_pagesize(space, vaddr);

WARNINGS

The page size for a given GVA can change over a period of time due to physical page demotion.

SEE ALSO

None
VASSERT(9F)

NAME
VASSERT - Test an assertion on kernels compiled with the OSDEBUG flag.

SYNOPSIS
#include <sys/debug.h>

VASSERT (expr);

PARAMETERS
expr An expression that evaluates to true or false.

DESCRIPTION
The VASSERT kernel function calls panic if the kernel was compiled with OSDEBUG on and if the expression expr is false. Otherwise, it does nothing.

RETURN VALUES
None

CONSTRAINTS
None

SEE ALSO
panic(9F)
vsnprintf(9F)

NAME
vsnprintf - Kernel version of vsnprintf.

SYNOPSIS
#include <sys/kern_svcs.h>
int vsnprintf(  
    char *str,  
    size_t maxsize,  
    const char *fmt,  
    va_list ap  
) ;

PARAMETERS
str Address of buffer to hold the formatted string.
maxsize Specifies the maximum number of characters to format.
fmt A set of printing characters and limited conversion specifications, as defined in
vsnprintf(3S).
ap A varargs argument list corresponding to a format conversion specification, as defined in
snprintf(3S).

DESCRIPTION
The vsnprintf kernel function is similar to the C library vsnprintf routine (see vsnprintf(3S)).
The vsnprintf writes information to the str array.
The kernel vsnprintf routine can accept the following formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Printed Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>%; no argument</td>
</tr>
<tr>
<td>%c</td>
<td>Character from integer argument</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal from integer argument</td>
</tr>
<tr>
<td>%lx</td>
<td>The argument x is long integer; x is one of x,d,u, or o.</td>
</tr>
<tr>
<td>%0</td>
<td>Unsigned octal from integer argument</td>
</tr>
<tr>
<td>%s</td>
<td>Characters from string argument</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal from integer argument</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal from integer argument</td>
</tr>
</tbody>
</table>

These formats are the same as in vsnprintf(3S). Other formats specified in vsnprintf(3S) are not supported.

RETURN VALUES
The vsnprintf function returns the length of formatted string.

CONSTRAINTS
None

SEE ALSO
msg_printf(9F),printf(9F),vsnprintf(3S)
waiting_in_select(9F)

NAME

waiting_in_select - Determines if a kernel thread is waiting on a select condition.

SYNOPSIS

#include <sys/select.h>

int waiting_in_select(
    struct kthread *threadp
);

PARAMETERS

threadp Pointer to a kernel thread.

DESCRIPTION

The waiting_in_select kernel function determines if the specified kernel thread threadp is waiting on a select condition. This function replaces accesses to the global variable selwait, which is no longer supported.

RETURN VALUES

The waiting_in_select function returns the following values:

0  Thread is not waiting on a select condition.
1  Thread is waiting on a select condition.

CONSTRAINTS

None

EXAMPLES

#include <sys/select.h>
#include <sys/kthread.h>

#define MYSEL_COLLISION 1 /* my select collision flag */

static int
mydev_select(dev_t dev, int rw)
{
    struct kthread * kthreadp;
    ...
    /*
    * Check if another kernel thread is already
    * sleeping on a select condition with the driver.
    */
    kthreadp = myselstruct->thread;
    if (kthreadp &
        waiting_in_select(kthreadp) {
            /*
            * There is going to be a collision of multiple
            * threads sleeping on a select condition.
            */
            myselstruct->selflag |= MYSEL_COLLISION;
        } else {
            /*
            * Save the kernel thread pointer of this
            * thread that will sleep on a select condition.
            */
            myselstruct->thread = kthreadp_self();
    }
}
static void
mydev_selwakeup(void)
{
    selwakeup(myselstruct->thread,
               myselstruct->selflag & MYSEL_COLLISION);
    myselstruct->thread = NULL;
    myselstruct->selfflag &= ~MYSEL_COLLISION;
}

SEE ALSO

selwakeup(9F)
This chapter contains reference pages describing routines and data structures used by drivers to communicate with the WSIO CDIO.

Some earlier interfaces have been deprecated. See Appendix A (page 1107) for a list.
block_to_raw(9F)

NAME

block_to_raw - Return the corresponding character dev_t for a given a block dev_t.

SYNOPSIS

#include <io/gio.h>

dev_t block_to_raw(
    dev_t dev
);

PARAMETERS

dev Block dev_t.

DESCRIPTION

The routine returns the character dev_t of a block dev_t. It also checks to ensure that the passed dev_t is a valid dev_t.

RETURN VALUES

On success, returns the Raw or character dev_t for the passed block dev_t. On error, returns the following:

NODEV The passed dev_t is not valid, or the drv_info is NULL.

CONSTRAINTS

None

SEE ALSO

raw_to_block(9F)
class_get_node(9F)

NAME

class_get_node - Get the I/O tree node associated with a class and instance number.

SYNOPSIS

#include <io/gio.h>
void *class_get_node(
    void *class,
    int  instance
);

PARAMETERS

class A pointer to the name of the associated class.
instance Instance number within the class of the node.

DESCRIPTION

The class_get_node routine returns the handle of the I/O node that is assigned the instance number in the given class for the specified class name and an instance number.

RETURN VALUES

If successful, returns an I/O tree node token. Otherwise, on error, returns NULL.

CONSTRAINTS

None

SEE ALSO
NAME

devsw_entry_t - Enumeration for selecting driver functions in the switch table.

SYNOPSIS

#include <io/conf.h>

STRUCTURE MEMBERS

enum devsw_entry {
    DEVSW_OPEN = 1,
    DEVSW_CLOSE,
    DEVSW_READ,
    DEVSW_WRITE,
    DEVSW_IOCTL,
    DEVSW_SELECT,
    DEVSW_OPTION1,
    DEVSW_STRATEGY,
    DEVSW_DUMP,
    DEVSW_PSIZE,
    DEVSW_PSIZE1,
} devsw_entry_t;

DESCRIPTION

The values of this enumerated type are passed to io_invoke_devsw to indicate which driver entry point to invoke in the driver switch table.

SEE ALSO

devsw_table_t(9S)
NAME
devsw_table_t - Enumeration to specify a driver switch table.

SYNOPSIS
#include <io/gio.h>

STRUCTURE MEMBERS
typedef enum devsw_table {
    CDEVSW_ENTRY 0
    BDEVSW_ENTRY 1
} devsw_table_t;

CDEVSW_ENTRY Character device driver switch table.
BDEVSW_ENTRY Block device driver switch table.

DESCRIPTION
This enumerated type is passed to io_invoke_devsw to indicate which switch table or to
io_get_devsw_len to indicate which device driver switch table to reference.

SEE ALSO
io_get_devsw_len(9F), io_invoke_devsw(9F).
NAME
dma_sync_IO - Synchronize processor and device views of memory.

SYNOPSIS

void dma_sync_IO(
    uint32_t addr_type,
    void *addr,
    int length,
    uint32_t hints
);

PARAMETERS

addr-type  Space ID corresponding to addr.
addr       Virtual address (processor view) of memory object.
length     Size of the memory object, in bytes, pointed to by addr.
hints      Bit-wise OR of hints that change the behavior of dma_sync. If no hints are given, the call results in a SYNC instruction on noncoherent platforms and a SYNCDMA on semicoherent platforms; nothing is done on fully coherent platforms. The defined hints are:

IO_ACCESSSED  Perform function only if the page has been accessed by a processor.
IO_MODIFIED   Perform function only if the page has been modified by a processor.
IO_NO_SYNC    Inhibit execution of SYNC or SYNCDMA instructions.
IO_PREFETCHED Perform function only if the processor prefetches data.
IO_READ       Purge processor caches for inbound data on noncoherent platforms. Done after completing the DMA data transfer.
IO_READ_START Purge processor caches for inbound data on noncoherent platforms and inhibit the SYNCDMA instruction on semicoherent platforms. Done prior to starting the DMA data transfer.
IO_SYNC_FORCPU Same as IO_READ.
IO_SYNC_FORDEV Same as IO_WRITE.
IO_SYNC_MEM   Synchronize processor caches with host memory; caches are flushed to memory when used with IO_WRITE (even on coherent platforms).
IO_WRITE      Flush processor caches for outbound data on noncoherent platforms.

DESCRIPTION

Drivers call dma_sync_IO to synchronize the processor caches with DMA transactions mastered by their devices. The dma_sync_IO function is sensitive to the underlying coherency of the platform. If the platform is coherent, dma_sync_IO does nothing; the hardware provides the coherency functionality. If the platform is semicoherent, dma_sync_IO handles the special case where the processor caches must be synchronized with data that have been read into host memory. If the platform is noncoherent, dma_sync_IO flushes (or purges and synchronizes the processor caches to maintain a consistent view of memory between processors and devices.
There are three cases to consider where drivers must call \texttt{dma\_sync\_IO}. These cases are prior to starting a write transaction, prior to starting a read transaction and after completing a read transaction:

1. Prior to starting a write transaction:
   For each buffer that is to be written out, the driver must call \texttt{dma\_sync\_IO} with the \texttt{IO\_WRITE} hint set. On noncoherent platforms, this will cause the associated processor caches to be flushed. For all but the last buffer, the \texttt{IO\_NO\_SYN} hint must also be set to reduce the performance penalty or synchronizing the cache flushes on noncoherent platforms.

2. Prior to start a read transaction:
   For each buffer that is to be read into, the driver must call \texttt{dma\_sync\_IO} with the \texttt{IO\_READ\_START} hint set. On noncoherent platforms, this will cause the associated processor caches to be purged. For all but the last buffers, the \texttt{IO\_NO\_SYNC} hint must also be set to reduce the performance penalty of synchronizing the cache purges on noncoherent platforms.

3. After completing a read transaction:
   For each buffer that has been read into, the drive must call \texttt{dma\_sync\_IO} with the \texttt{IO\_READ} hint set. On noncoherent platforms, this will cause the associated processor caches to be purged of data that may have been prefetched. For all but the last buffer, the \texttt{IO\_NO\_SYNC} hint must also be set to reduce the performance penalty of synchronizing the cache purges on noncoherent platforms. On semicoherent platforms, the processor caches will be made to synchronize with the data read when the \texttt{IO\_NO\_SYNC} hint is not set.

**CONSTRAINTS**

None

**SEE ALSO**
NAME

driver_addr_probe - Interface driver address probing function.

SYNOPSIS

#include <sys/wsio.h>
#include <sys/ioparams.h>

int driver_addr_probe (  
    void *this_node,  
    int (*dev_probe) (),  
    drv_info_t *drv_info,  
    void *probe_id,  
    hw_path_t *hw_path,  
    struct ics_table_type *isc,  
    int probe_type,  
    char *name,  
    char *desc  
);  

PARAMETERS

this_node A pointer to an io_tree_node structure.
dev_probe Probe function registered by device driver to be called by driver_addr_probe.
drv_info The drv_info_t structure registered with wsio_install.
probe_id A unique identifier (for example, first 4 bytes of SCSI Inquiry data).
hw_path A pointer to a structure containing the hardware path information of the module being probed.
isc A pointer to the ISC structure assigned to the interface node that is being probed.
probe_type The type of hardware probe to perform.
    Defined types are:
    PROBE_FIRST Start at first available address.
    PROBE_NEXT Increment the last address and start looking from there.
    PROBE_ADDRESS Look only for this address.
name A string describing the device.
desc A string describing the device found by the probe (usually 8 bytes of Vendor ID followed by 16 bytes of Product ID).

DESCRIPTION

The driver_addr_probe WSIO function is provided by the driver writer; it can have any unique name. Register this function with WSIO by executing the wsio_register_addr_probe routine as part of the interface driver_attach routine. Commonly, driver is replaced by your driver's name.

See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES

PROBE_SUCCESS Successfully found something it identified.
PROBE_UNSUCCESSFUL Got to end of the appropriate address range without finding anything, or something went wrong with the probe.
CONSTRANTS
None

SEE ALSO

driver_class_probe(9E), wsio_probe_dev_info(9F), wsio_register_probe_func(9F),
wsio_register_addr_probe(9F)
**NAME**

driver_attach - Claim a device for a driver.

**SYNOPSIS**

```c
int driver_attach(
    uint32_t card_id,
    struct ics_table_type *isc);
```

**PARAMETERS**

- `card_id` A four-byte card or product identifier.
- `isc` A pointer to the ISC structure assigned to the interface node that is being probed.

**DESCRIPTION**

The `driver_attach` WSIO function is provided by the driver writer; it can have any unique name. Pass the name to WSIO Services by including it in an attach chain with the driver's `driver_install` routine. Commonly, `driver` is replaced by the driver's name.

The kernel searches the I/O backplane for hardware. When it finds a device, it first does preliminary initialization. Then, it calls the `driver_attach` routine at the head of the corresponding attach chain (for example, `eisa_attach`) that was created by the `driver_install` routine.

Each `driver_attach` routine in the chain looks at the `card_id`. If it recognizes the device as its own, it claims the device with the `isc_claim` function, optionally puts a pointer to its `driver_if_init` routine in `isc->gfsw->init`, and performs any other appropriate initialization. Then, whether it claims the device or not, it passes the same parameters to the next routine in the chain, using the routine name it saved in the `driver_install` routine.

Since `isc_claim` sets the INITIALIZED flag in `isc->if_info->flags`, test this flag to see if there was a prior claim.

See the HP-UX 11i v3 Driver Development Guide for information.

**RETURN VALUES**

Each `driver_attach` routine is expected to return the return value returned by the next `driver_attach` routine in the chain. The end-of-chain function returns a unique completion code.

**CONSTRAINTS**

None

**EXAMPLE**

Be very careful with the `card_id` parameter. It is tempting to just define it as `PCI_ID` in a PCI `driver_attach` routine. Due to 64-bit kernel parameter passing conventions, use code similar to the following example in determining that the device is the drivers':

```c
int zzz_attach(uint32_t idparm, struct ics_table_type *isc)
{
    PCI_ID *id = (PCI_ID *)&idparm;
    if (!(id->vendor_id != MY_VENDOR_ID&&
        (id->device_ID != MY_DEV_ID)))
        return (my_saved_attach(idparm, isc);
    else {
        /* code to claim card - set up isr, etc. */
```
return (my_saved_attach(idparm, isc);
}

SEE ALSO

get_new_isc(9F), isc_table_type(9F)
NAME

driver_close - Close a device.

SYNOPSIS

#include <sys/conf.h>

int driver_close(
    dev_t dev,
    int flag,
    int mode
);

PARAMETERS

dev    The device number of the file to be closed. The driver_close routine can extract the
       major and minor numbers from the device number (see major(9F) and minor(9F)).
       (A user process specifies a file descriptor in the close system call.)

flag   A value corresponding to the flag field in the driver_open call. See driver_open(9E).

mode   Determines whether this is a call to a block or char driver. This parameter is not accessible
       from a close call.

DESCRIPTION

The driver_close WSIO function is provided by the driver writer. It can have any unique
name. Pass the name to WSIO Services by specifying it in the d_close field of the drv_ops
structure. Commonly, driver is replaced by the driver’s name.

See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES

The file system function which calls the driver through the bdevsw or cdevsw tables always
returns success (0) to the higher level file system function which called it, ignoring the return
value it gets from the driver.

Therefore, the driver_close routine need not return a valid value. However, to avoid problems
(as with strict compiler return value checking), the driver_close routine must return some
integer value.

CONSTRAINTS

None

SEE ALSO

close(2), driver_open(9E), drv_ops(9S), open(2)
NAME

driver_dev_init – Initialize a device driver.

SYNOPSIS

int driver_dev_init(void);

PARAMETERS

None

DESCRIPTION

The driver_dev_init WSIO function is provided by the driver writer. It can have any unique name. Pass the name to WSIO Services by including it in the init chain, dev_init, with the driver_install routine. Commonly, driver is replaced by the driver's name.

See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES

Each driver_dev_init routine is expected to return the return value returned by the next driver_dev_init routine in the chain. The end-of-chain function returns a unique completion code.

CONSTRAINTS

None

SEE ALSO

driver_install(9E)
NAME

driver_dev_probe - Interface driver device probing function.

SYNOPSIS

#include <sys/wsio.h>
#include <sys/ioparams.h>

int driver_dev_probe (
    void *this_node,
    drv_info_t *drv_info,
    void *probe_id,
    hw_path_t *hw_path,
    struct_ics_table_type *isc,
    int probe_type,
    char *name,
    char *desc
);

PARAMETERS

tthis_node A pointer to an io_tree_node structure.
drv_info The drv_info_t structure registered with wsio_install.
probe_id A unique identifier (for example, first 4 bytes of SCSI Inquiry data).
hw_path A pointer to a structure containing the hardware path information of the module being probed.
isc A pointer to the ISC structure assigned to the interface node that is being probed.
probe_type The type of hardware probe to perform.

Defined types are:

PROBE_FIRST Start at first available address.
PROBE_NEXT Increment the last address and start looking from there.
PROBE_ADDRESS Look only for this address.

name A string describing the class of the device.
desc A string describing the device found by the probe (usually 8 bytes of Vendor ID followed by 16 bytes of Product Id).

DESCRIPTION

The driver_dev_probe WSIO function is provided by the driver writer. It can have any unique name. If one is needed, register it with WSIO by executing the wsio_register_dev_probe routine as part of the driver_install routine. Commonly, driver is replaced by your driver's name.

See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES

PROBE_SUCCESS Successfully found something it can identify.
PROBE_UNSUCCESSFUL Got to end of the appropriate address range without finding anything, or something went wrong with the probe.

CONSTRAINTS

None
SEE ALSO

driver_addr_probe(9E), wsio_register_addr_probe(9F), wsio_probe_dev_info(9F),
wsio_register_dev_func(9F)
driver_if_init(9E)

NAME
driver_if_init -- Initialize interface driver.

SYNOPSIS
#include <sys/io.h>

int driver_if_init(
    struct isc_table_type *isc
);

PARAMETERS
isc Pointer to an ISC structure for an interface this driver controls.

DESCRIPTION
The driver_if_init WSIO function is provided by the driver writer. It can have any unique name. Pass the name to WSIO Services by specifying it in the isc->gfsw->init of the ISC structure with driver's driver_attach routine. Commonly, driver is replaced by the driver's name.

See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES
0 Successful completion.
-1 Error

CONSTRAINTS
None

SEE ALSO
driver_attach(9E), isc_table_type(9F),
driver_install(9E)

NAME

driver_install - Register a driver with the system.

SYNOPSIS

void driver_install(void);

PARAMETERS

None

DESCRIPTION

The driver_install WSIO function is provided by the driver writer. The name must be in the format shown, with driver replaced by the name of the driver.
See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES

None. If the routine encounters an error, it must print an error message and return.

CONSTRAINTS

None

SEE ALSO

cfg(1M), driver_attach(9E), driver_dev_init(9E), driver_install(9E), driver_probe(9E), install_driver(9S), master(4), wsio_install_driver(9F), wsio_register_addr_probe(9F), wsio_register_probe_func(9F)
NAME

driver_ioctl - Execute driver-specific control functions.

SYNOPSIS

#include <sys/conf.h>

int driver_ioctl(
    dev_t dev,
    int cmd,
    caddr_t arg_ptr,
    int flag
);

PARAMETERS

dev Device number.

 cmd Command word.

 arg_ptr Pointer to the command word arguments, if any.

 flag File access flags.

DESCRIPTION

The driver_ioctl WSIO function is provided by the driver writer. It can have any unique
name. Pass the name to WSIO Services by specifying it in the d_ioctl field of the drv_ops
structure. Commonly, driver is replaced by the driver's name.

See the HP-UX 11i v3 Driver Development Guide for information.

0 Successful completion.

<> 0 Error. The value is expected to be an error value.

CONSTRAINTS

None

LP64 CONSIDERATIONS

Pay particular attention to the cmd parameter, which has different values depending upon the
calling program environment. See the HP-UX 11i v3 Driver Development Guide for more
information.

SEE ALSO

drv_ops(9S), errno(2), ioctl(2), ioctl(5)
driver_isr(9E)

NAME

driver_isr - Execute device interrupt in interrupt context.

SYNOPSIS

int driver_isr(
    uintptr_t arg
);

PARAMETERS

arg Parameter passed in with the wsio_intr_alloc call.

DESCRIPTION

The driver_isr WSIO function is provided by the driver writer, it can have any unique name. Pass the name to WSIO Services by specifying it as a parameter of the wsio_intrAlloc function, executed in the driver_attach or driver_if_init routine. Commonly, driver is replaced by the driver's name.

See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES

0 The card does not belong to this driver.
1 This routine handled the interrupt.

CONSTRAINTS

None

SEE ALSO

wsio_intr_alloc(9F)
driver_load(9E)

NAME
driver_load - Called by the DLKM infrastructure while loading the driver dynamically.

SYNOPSIS

int driver_load(
    void *drv_infop
);

PARAMETERS

drv_infop A pointer to the drv_info structure for the driver.

DESCRIPTION

A pointer to the driver_load function is passed to the DLKM infrastructure through
modwrapper structure. The name of the driver_load function must be driver_load.
The driver_load function must perform all the functions that are performed by the
driver_install function except for the following differences:

- The driver_load function for a PCI interface driver must call
  mod_wsio_attach_list_add to add its attach function, if any, to the “DLKM attach list”
  instead of inserting its driver_attach at the head of the global pci_attach chain.
- The driver_load function must perform any driver specific initialization and must not
  add its driver_init function to the dev_init chain.
- Call wsio_activate_probe to activate any probe routine registered in the load routine.

NOTE: Add the static keyword at the end of initfunc driver_install directive in the
modmeta file to prevent the kernel configuration from calling both driver_install and
driver_load() routines.

RETURN VALUES

On success, driver_load returns 0, otherwise it returns an error number from <sys/errno.h>
file to indicate an error.

EXAMPLES

The following is an example of a WSIO interface driver load function:

```c
#include <sys/wsio.h>
#include <sys/errno.h>
#include <sys/moddefs.h>

static wsio_drv_info_t driver_wsio_info = { ... };
static int driver_pci_attach (...);
static void driver_init(...);
int driver_load(void *arg)
{
    /* Pass the drv_info passed to this function to wsio_install_driver */
    driver_wsio_info.drv_info = (drv_info_t *) arg;

    /* Register the driver with WSIO */
    if (!wsio_install_driver(&driver_wsio_info)) {
        printf("<driver>: wsio_install_driver failed!!\n");
        goto err3;
    }
    /* Perform driver-specific initialization, but do not call the next
    * function in the dev_init list
    */
```
(void) driver_init();
/* Add the attach function to the DLKM attach list */
if (mod_wsio_attach_list_add(MOD_WSIO_PCI, &driver_pci_attach,
   DRIVER_NAME) != 0) {
    printf("<driver>: mod_wsio_attach_list_add failed!!\n");
    goto err2;
}
/* Register the device probe */
if (wsio_register_dev_probe(IF_CLASS, driver_probe, "probe_name")
    != 0) {
    printf("<driver>: wsio_register_dev_probe failed!!\n");
    goto err1;
}
/* Attach the probe function to the drv_info structure */
if (wsio_activate_probe("probe_name", driver_wsio_info.drv_info)
    != 0 ) {
    printf("<driver>: wsio_activate_probe failed!!\n");
    (void) wsio_unregister_probe(IF_CLASS, "probe_name");
    goto err1;
}
return (0);
err1:
    (void) mod_wsio_attach_list_remove(MOD_WSIO_PCI, &driver_pci_attach);
err2:
    (void) wsio_unregister_driver(&driver_wsio_info);
err3:
    return (ENXIO);
}

SEE ALSO

driver_install(9E), driver_unload(9E), modwrapper(9F), kcmodule(1M), wsio_install_driver(9F),
wsio_uninstall_driver(9F), driver_init(9E), mod_wsio_attach_list_add(9F),
mod_wsio_attach_list_remove(9F), wsio_register_dev_probe(9F), wsio_activate_probe(9F),
wsio_unregister_probe(9F), modmeta(4)
driver_unload(9E)

NAME
driver_unload - Called by the DLKM Infrastructure during unload of a dynamically unloadable
driver

SYNOPSIS

int driver_unload(
    void *drv_infop
);

PARAMETERS
drv_infop A pointer to the drv_info structure for the driver.

DESCRIPTION

A pointer to the driver_unload function is passed to the DLKM infrastructure through
modwrapper structure. The name of the driver_unload function must be driver_unload.
A driver that declares support for the unload capability in its metadata must provide a
driver_unload function.

For the driver to be unloaded, the number of opens on the device file associated with the driver
must be zero. The driver_unload function does not need to check for this. The driver's unload
routine is not called if there are any outstanding opens on the device special file.

The driver_unload function must call mod_wsio_attach_list_remove to remove its
attach function, if its attach function was added to the DLKM attach list in the driver_load
function. It must call the wsio_unregister_dev_probe to unregister the driver's probe
function, if the probe function was registered with WSIO during the driver_load. The
driver_unload function must perform any other necessary driver-specific cleanup operations
(for example, clear any outstanding timeouts, free any interrupt objects, unregister ISR routine,
free allocated memory resources, free spinlocks, and call ktune_unregister_handler to
unregister any tunable handlers that were registered by the tunable_init function). It must call
wsio_uninstall_driver passing to it a pointer to the driver's wsio_drv_info structure.

If in any step, the driver_unload fails, the driver must undo any action prior to the failure
and return an appropriate error from <sys/errno.h>.

RETURN VALUES

On success, driver_unload returns 0; otherwise, it returns an error number from <sys/
errno.h> file to indicate an error.

EXAMPLES

The following is an example of a WSIO interface driver unload function:

```c
#include <sys/wsio.h>
#include <sys/errno.h>
#include <sys/moddefs.h>

static wsio_drv_info_t driver_wsio_info = { ... };
static int driver_pci_attach (...);

int driver_unload(void *arg)
{
    driver_wsio_info.drv_info = (drv_info_t *) arg;
    /* Remove the attach function from the DLKM attach list */
    if (mod_wsio_attach_list_remove(MOD_WSIO_PCI, &driver_pci_attach)
        != 0) {
        printf("<driver>: mod_wsio_attach_list_remove failed!!\n");
        return (ENXIO);
    }
    /* Unregister the device probe */
    if (wsio_unregister_dev_probe(IF_CLASS, driver_probe, "probe_name")
        != 0) {
        printf("<driver>: wsio_unregister_dev_probe failed!!\n");
    }
    /* Other cleanup operations */
    ...
    return 0;
}
```
mod_wsio_attach_list_add(MOD_WSIO_PCI, driver_pci_attach,
   DRIVER_NAME);

return (ENXIO);

/* Perform driver cleanup for each driver instance, find the isc claimed
 * by the driver, clear any pending timeouts, unregister ISR routine, free
 * allocated memory and resources, call ktune_unregister_handler to
 * unregister any handlers that were registered by tunable_init function
 */
if (wsio_uninstall_driver(&driver_wsio_info) != 0) {
    printf("<driver>: wsio_uninstall_driver failed!!\n");
    mod_wsio_attach_list_add(MOD_WSIO_PCI, driver_pci_attach,
       DRIVER_NAME);
    wsio_register_dev_probe(IF_CLASS, driver_probe, "probe_name");
    return (ENXIO);
}
return 0;

SEE ALSO

driver_install(9E), driver_load(1M), modwrapper(1M), kcmodule(1M), wsio_install_driver(9F),
wsio_uninstall_driver(9F), driver_init(9E), mod_wsio_attach_list_add(9F),
mod_wsio_attach_list_remove(9F), wsio_register_dev_probe(9F), wsio_activate_probe(9F),
wsio_unregister_probe(9F)
**NAME**

driver_minor_build -- Build a minor number.

**SYNOPSIS**

```
#include <sys/wsio.h>
#include <sys/ioparams.h>

int driver_minor_build(
    isc_table_type *isc,
    hw_path_t *dev_path,
    char *option
);
```

**PARAMETERS**

- `isc` A pointer to the ISC structure associated with the interface card for the device.
- `dev_path` A pointer to a structure containing device hardware path information relative to the interface card.
- `option` A null-terminated string indicating device-specific options.

**DESCRIPTION**

The `driver_minor_build` WSIO function is provided by the driver writer, it can have any unique name. Pass the name to WSIO Services by specifying it in the `drv_minor_build` field of the `wsio_drv_data_t` structure. Commonly, `driver` is replaced by the driver's name.

See the *HP-UX 11i v3 Driver Development Guide* for information.

**RETURN VALUES**

- `>0` Successful completion. The value is the minor number.
- `-1` Error. The minor number could not be built.

**CONSTRAINTS**

None

**SEE ALSO**

`wsio_drv_data_t(9F)`
**NAME**

`driver_minphys` - Driver specific transfer size adjustment.

**SYNOPSIS**

```c
#include <sys/buf.h>
void driver_minphys(
    struct buf *bp
);
```

**PARAMETERS**

`bp` Transfer information structure.

**DESCRIPTION**

The `driver_minphys` WSIO function is provided by the driver writer, it can have any unique name. Pass the name to `physio` by specifying it in the `mincnt` parameter in the call to `physio`. Commonly, `driver` is replaced by your driver's name.

The `driver_minphys` function adjusts the `bp->b_bcount` field of the `buf` structure passed in.

**RETURN VALUES**

`driver_minphys` is a `void` function.

**EXAMPLES**

The following example illustrates a `driver_minphys` routine for a device that can handle at most `NBPG` size transfers:

```c
void mydriver_minphys(struct buf *bp) {
    if (bp->b_bcount > NBPG)
        bp->b_bcount = NBPG;
}
```

**CONSTRAINTS**

None

**SEE ALSO**

`minphys(9F), physio(9F)`
NAME

driver_open - Open a device.

SYNOPSIS

#include <sys/conf.h>

int driver_open(
    dev_t dev,
    int oflags,
    intptr_t dev,
    int mode
);

PARAMETERS

dev The dev_t device number of the file to be opened. The driver_open routine can extract the major and minor numbers from the device number. See major(9F) and minor(9F).

oflags A value corresponding to the oflag parameter of the open system call. The kernel executes the oflag functions (described in fcntl(5) and open(2)) before it calls the driver. The driver, therefore, can usually ignore these flags. Nevertheless, the kernel translates the O_xxxx values into corresponding F_xxxx values, which it passes to the driver_open routine. The flags of possible interest to the driver include: FREAD, FWRITE, FNDELAY, and FEXCL.

mode Whether this is a call to a block or char driver. This parameter is not accessible from an open(2) call.

dummy A parameter used as by some drivers, though it is not accessible from an open(2) call.

DESCRIPTION

The driver_open WSIO function is provided by the driver writer; it can have any unique name. Pass the name to WSIO Services by specifying it in the driver_open field of the drv_ops structure. Commonly, driver is replaced by the driver’s name.

See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES

driver_open is expected to return the following values:

0 Successful completion.
<0 Error. The value is expected to be an errno value.

If the driver_open routine is successful, the kernel’s open call returns a file descriptor to the user. If it is unsuccessful, the kernel returns -1 to the user and sets errno to the value returned by the driver_open routine. The user’s process can check the returned value and errno to determine whether an error occurred. See the <errno.h> header file for possible values for errno.

The driver_open routine must return an error under these conditions. See open(2) for the expected error names:

• The device is off line.
• The device does not exist.
• The device was never configured into the system.
• The initialization of the device failed.
• The device is an exclusive-open device, and it is already open.

**CONSTRAINTS**

None

**SEE ALSO**

`drv_ops(9S), open(2)`
NAME
driver_psize - Get swap partition size of a device.

SYNOPSIS
#include <sys/conf.h>
int driver_psize(
    dev_t dev
);

PARAMETERS
dev    Contains encoded major and minor numbers.

DESCRIPTION
The driver_psize WSIO function is provided by the driver writer; it can have any unique
name. Pass the name to WSIO Services by specifying it in the d_psize field of the drv_ops
structure. Commonly, driver is replaced by your driver's name.

The driver_psize function must return the size of the swap partition on a block swapping
device. It is called by the kernel. Consider writing this routine only if your device is used for
swapping.

See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES
>0    Successful completion. The value is the swap partition size.
-1    Error

CONSTRAINTS
None

SEE ALSO
drv_ops(9S)
driver_read(9E)

NAME

driver_read – Read data from and to a character device.

SYNOPSIS

#include <sys/conf.h>

int driver_read(
    dev_t dev,
    struct uio *uio
);

PARAMETERS

dev  The device number of the associated device file. The routine can extract the major and
     minor numbers from the device number. The driver_open routine must verify that the
     minor number is valid.

uio  A pointer to a uio structure. The uio structure contains information about the data being
     read or written.

DESCRIPTION

When a user process issues a read, readv, write, or writev system call for a character device,
the kernel puts information about the request in the uio and iovec structures and dispatches
control to the driver_read or driver_write routine for that device, passing the uio structure
to the driver as a parameter.

See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES

0    Successful completion.

<0   Error. The value is expected to be an errno value.

CONSTRAINTS

None

EXAMPLES

See physio(9F) and uiomove(9F).

SEE ALSO

drv_ops(9S), physio(9F), uiomove(9F)
driver_select(9E)

NAME
driver_select - Test I/O completion on a device.

SYNOPSIS
#include <sys/conf.h>
int driver_select(
    dev_t dev,
    int flag
);

PARAMETERS
dev  The device number.
flag  The type of readiness to test, according to the following values:
   FREAD  Read
   FWRITE  Write
   0  Exception conditions

DESCRIPTION
The driver_select WSIO function is provided by the driver writer; it can have any unique name. Pass the name to WSIO Services by specifying it in the d_select field of the drv_ops structure. Commonly, driver is replaced by your driver's name.
See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES
<>0  True — The device or driver is ready for read or write or an exception condition was found. The kernel sets the corresponding bit in the bit-mask field that driver_select returns to the user.
0  False — The device or driver is not ready for read or write or no exception condition was found. The driver_select function puts the calling process to sleep until the condition becomes true. The driver must inform the system when this condition becomes true.
If the driver_select routine detects an error while selecting for read or write, it must return false and set an error in u.u_error. If it detects an error while selecting for an exception condition, it must return true and set an error in u.u_error.

CONSTRAINTS
None

SEE ALSO
drv_ops(9S), selwakeup(9F), select(2)
driver_strategy(9E)

NAME
driver_strategy - Execute block read or write for character or block devices.

SYNOPSIS
#include <sys/conf.h>
int driver_strategy(
    struct buf *bp
);

PARAMETERS
bp Pointer to a buf structure.

DESCRIPTION
The driver_strategy WSIO function is provided by the driver writer; it can have any unique name. For a block device, pass the name to WSIO Services by specifying it in the driver_strategy field of the drv_ops structure. For a character device, pass the name as a parameter to physio. Commonly, driver is replaced by your driver's name.

See he HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES
None

CONSTRAINTS
None

SEE ALSO
physio(9F)
**NAME**

`driver_write` - Write data from/to a character device.

**SYNOPSIS**

```c
#include <sys/conf.h>

int driver_write(
    dev_t dev,
    struct uio *uio
);
```

**PARAMETERS**

- `dev` The device number of the associated device file. The routine can extract the major and minor numbers from the device number. The `driver_open` routine must verify that the minor number is valid.
- `uio` A pointer to a `uio` structure. The `uio` structure contains information about the data being read or written.

**DESCRIPTION**

When a user process issues a `read`, `readv`, `write`, or `writev` system call for a character device, the kernel puts information about the request in the `uio` and `iovec` structures and dispatches control to the `driver_read` or `driver_write` routine for that device, passing the `uio` structure to the driver as a parameter.

See the HP-UX 11i v3 Driver Development Guide for information.

**RETURN VALUES**

- `0` Successful completion.
- `<0` Error. The value is expected to be an `errno` value.

**CONSTRAINTS**

None

**SEE ALSO**

`drv_ops(9S), physio(9F), uiomove(9F)`
NAME
drv_info - Driver information structure.

SYNOPSIS
#include <sys/conf.h>

PARAMETERS
typedef struct drv_info
{
    char      *name;       /* Name of driver */
    char      *class;      /* Device class (see below) */
    ubit32    flags;       /* Device flags (see below) */
    int       b_major;     /* Block device major number */
    int       c_major;     /* Character device major number */
    cdio_t    *cdio;       /* Drivers set this to NULL */
    void      *gio_private; /* Drivers set this to NULL */
    void      *CDIO_PRIVATE; /* Drivers set this to NULL */
} drv_info_t;

DESCRIPTION
All CDIOs use the driver-specific fields in the drv_info_t CDIO structure type, defined in <sys/conf.h>, to describe certain parameters on the driver. A drv_info_t structure must be statically allocated.
The relevant fields are described. All other fields in a drv_info_t must be NULL.

STRUCTURE MEMBERS

<table>
<thead>
<tr>
<th>Field</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Pointer to a string containing the name of the driver. This is the name used in the system file (usually /stand/system), in the $DRIVER_INSTALL section of a master file in /usr/conf/master.d, and a the value for driver in the driver_install function name. See config(1M) and master(4). The current implementation of kernel functions that access name require that the string be less than 16 characters long.</td>
</tr>
<tr>
<td>class</td>
<td>Pointer to a string containing the name of the class that the driver is in. Interface drivers typically use ext_bus. Device drivers use classes that describe the general type of device (for example, disk, type, and pseudo). For interface drivers, instances of a card are enumerated within each class as they are identified by the kernel at boot time</td>
</tr>
<tr>
<td>flags</td>
<td>The bit-wise OR of flag values that describe the driver, taken from:</td>
</tr>
<tr>
<td>DRV_CHAR</td>
<td>Character device driver.</td>
</tr>
<tr>
<td>DRV_BLOCK</td>
<td>Block device driver.</td>
</tr>
<tr>
<td>DRV_PSEUDO</td>
<td>Pseudo driver.</td>
</tr>
<tr>
<td>DRV_SCAN</td>
<td>Driver supports bus scanning</td>
</tr>
<tr>
<td>DRV_MP_SAFE</td>
<td>Driver provides its own multiprocessing protection. This flag and C_MGR_IS_MP in drv_ops_t must be consistent or the kernel services will treat the driver as if it were not MPSAFE.</td>
</tr>
</tbody>
</table>
DRV_SAVECONF  Save configuration information to /etc/ioconfig. This file retains potentially volatile information, such as dynamic major numbers and card instance numbers, across reboots.

b_major  The major number if this is a block device. Set it to -1 for dynamic assignment or if it is not a block device.

c_major  The major number if this is a character device. Set it to -1 for dynamic assignment or if it is not a character device.

NOTE: The values specified for b_major and c_major override the values entered in a master file in /usr/conf/master.d (see master(4)).

SEE ALSO

config(1M), driver_install(9E), drv_ops(9S), wsio_drv_info(9F)
drv_ops(9S)

NAME
drv_ops - Structure to specify driver entry points.

SYNOPSIS
#include <sys/conf.h>

PARAMETERS
typedef struct drv_ops
{
    int (*d_open) ();       /* block and character */
    int (*d_close) ();      /* block and character */
    int (*d_strategy) ();   /* block */
    int (*d_dump) ();       /* NULL (obsolete) */
    int (*d_psize) ();      /* block */
    int (*reserved0) ();    /* NULL */
    int (*d_read) ();       /* character */
    int (*d_write) ();      /* character */
    int (*d_ioctl) ();      /* character */
    int (*d_select) ();     /* character */
    int (*d_option1) ();    /* NULL */
    pfilter_t *pfilter;     /* block and character */
    d_psize1_t (*d_psize1); /* block */
    int (*_drv_cb)(drv_cb_opcode_t,uintptr_t *); /* block and character */
    int (*reserved3) ();    /* NULL */
    int d_flags;            /* block and character */
} drv_ops_t;

DESCRIPTION
The drv_ops_t CDIO structure type, defined in <sys/conf.h>, contains pointers to all driver entry points. A drv_ops_t structure must be statically allocated.

The relevant fields are described in the STRUCTURE MEMBERS section. All other fields in drv_ops_t must be NULL. Except as noted, entry points that do not apply to the driver or that the driver does not provide must be NULL (for example, d_read has no meaning for a printer).

STRUCTURE MEMBERS
The Device Type column indicates whether the field applies to character-only, block-only, or both types of drivers as shown in Table 2-1 “Device Driver Fields” for drv_ops_t structure types.

Table 2-1 Device Driver Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Device Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_open</td>
<td>both</td>
<td>Pointer to the driver_open routine, which enables a device for subsequent operations.</td>
</tr>
<tr>
<td>d_close</td>
<td>both</td>
<td>Pointer to the driver_close routine, which performs the tasks required when a device is closed.</td>
</tr>
<tr>
<td>d_strategy</td>
<td>block</td>
<td>Pointer to the driver_strategy routine, which queues I/O requests for either reading or writing.</td>
</tr>
<tr>
<td>d_psize</td>
<td>block</td>
<td>Pointer to the driver_psize routine. For a swapping device, it must return the size of the swap partition.</td>
</tr>
</tbody>
</table>
Table 2-1 Device Driver Fields (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Device Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_read</td>
<td>character</td>
<td>Pointer to the driver_read routine, which must return the requested data transferred from the device.</td>
</tr>
<tr>
<td>d_write</td>
<td>character</td>
<td>Pointer to the driver_write routine, which must write the requested data to the device.</td>
</tr>
<tr>
<td>d_ioctl</td>
<td>character</td>
<td>Pointer to the driver_ioctl routine, which sends control information to, or gets it from a device.</td>
</tr>
<tr>
<td>d_select</td>
<td>character</td>
<td>Pointer to the driver_select routine, which can be used to test for I/O completion and driver-dependent exception conditions. If the device is always ready for reading or writing, put seltrue in the d_select field. If this happens, calls to select always return true without invoking the driver.</td>
</tr>
<tr>
<td>pfilter</td>
<td>both</td>
<td>Pointer to a pfilter_t structure, use the &amp;cpd_pfilter pointer. This structure provides backward compatible routines for disk structures with fixed partitions, such as the Series 800 computers before the availability of the Logical Volume Manager (LVM). The &amp;cpd_pfilter pointer is required for such disks; it is ignored under other conditions (or use NULL).</td>
</tr>
<tr>
<td>d_psizel</td>
<td>block</td>
<td>For a given minor number, it returns the 64-bit value of the partition size. The current d_psizel_t is maintained for backward compatibility, but only returns a 32-bit value. This entry point is defined as follows in conf.h: [ \text{int (*d_psizel_t)} (\text{dev_t dev, int64_t *size}); ] The difference between this and the current d_psizel_t entry point is that the former returns the size in the size parameter and the latter returns size as the function return value.</td>
</tr>
<tr>
<td>d_drv_cb</td>
<td>both</td>
<td>The driver callback routine is a new entry point into the driver. It is called with an opcode. Currently, the following opcodes are defined: * CB_DEV_2_NODE — Return the associated I/O tree node handle for a specified dev_t. * CB_GET_OPTS — Return the associated device-specific options for a specified dev_t. For more information, see the “Writing a Driver Callback Routine” section in the HP-UX 11i v3 Driver Development Guide.</td>
</tr>
<tr>
<td>d_flags</td>
<td>both</td>
<td>The bit-wise OR of flag values that indicate special features of the device. The flags give information about the device to the kernel. Drivers receive this information, but usually only validate it. Use 0 if no flags are set. The flag bit defines for d_flags are:</td>
</tr>
</tbody>
</table>

**C_ALLCLOSES** Force a call to driver_close on every closing of the device. (The default action is to call the driver’s close routine only on the last close of the device.)

**C_NODELAY** Tell the kernel to not wait for a write request to complete on this device. The default action is to wait for a write request to complete before returning control to the calling process.

**C_MGR_IS_MP** Identify the driver as safe for use in a multiprocessing environment. The flag and the DRV_MP_SAFE flag in drv_info_t must be consistent or the kernel services will treat the driver as if it were not MP SAFE.

**C_MAP_BUFFER_TO_KERNEL** Identify that the device driver needs physio to remap a user buffer to kernel space prior to calling the driver_strategy routine. This also identifies that after the associated buf structure has been marked iodone, physio will remap the buffer to user space.
C_OPAQ_DEV

Indicate that the device driver's minor number is opaque. The minor number does not encode any device information such as controller instance, target, LUN or device options. Drivers defining this flag should register for the CB_DEV_2_NODE and CB_GET_OPTS callbacks. This flag is mandatory for all mass storage interface drivers.

SEE ALSO

driver_close(9E), driver_ioctl(9E), driver_open(9E), driver_psize(9E), driver_read(9E), driver_select(9E),
driver_strategy(9E), driver_write(9E), wsio_drv_info(9E), drv_info(9S), physio(9F), select(2)
NAME
free_isc - Free a driver's ISC entry.

SYNOPSIS
#include <sys/wsio.h>
int free_isc(
    struct isc_table_type *isc
);  
PARAMETERS
isc  Pointer to an ISC entry.

DESCRIPTION
The free_isc kernel function frees an ISC entry that was obtained explicitly by a driver by using get_new_isc or another similar service. Call free_isc after a severe driver-disabling error or before a driver is unloaded.

RETURN VALUES
0  Successful completion.
-1  Error

CONSTRAINTS
None

SEE ALSO
get_new_isc(9F)
NAME
get_new_isc - Allocate a new ISC structure for this card function.

SYNOPSIS
#include <sys/io.h>
struct isc_table_type *get_new_isc(
    struct isc_table_type *dd_isc
);

PARAMETERS
dd_isc Pointer to a currently allocated ISC structure.

RETURN VALUES
None

DESCRIPTION
The get_new_isc WSIO function allocates a new ISC structure when one more is needed, like a multifunction card.

If the isc->ftn_no field is not -1 in an entry for a multifunction card, the driver_attach routine must call get_new_isc to allocate a new ISC structure for the driver’s functions and set the isc->ftn_no field of the new ISC structure to the function number for its portion of the card, and then continue its normal power-on initializations, using the new ISC structure that was returned from get_new_isc.

Pass the new ISC on to the next driver in the attach chain.

The get_new_isc function allocates and zeros out a new ISC structure and then performs the following:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus_type</td>
<td>Copied from old_isc</td>
</tr>
<tr>
<td>my_isc</td>
<td>Copied from old_isc</td>
</tr>
<tr>
<td>if_reg_ptr</td>
<td>Copied from old_isc</td>
</tr>
<tr>
<td>bus_info</td>
<td>Copied from old_isc</td>
</tr>
<tr>
<td>ftn_no</td>
<td>Set to -1, the caller must correctly set this field after call</td>
</tr>
<tr>
<td>old_isc-&gt;next_ftn</td>
<td>Set to the new isc</td>
</tr>
<tr>
<td>if_info</td>
<td>Allocated and then copied from old_isc</td>
</tr>
<tr>
<td>new-&gt;next_ftn</td>
<td>Set to NULL</td>
</tr>
<tr>
<td>ifsw</td>
<td>Copied from old_isc</td>
</tr>
<tr>
<td>if_drv_data</td>
<td>Copied from old_isc</td>
</tr>
<tr>
<td>gfsw</td>
<td>Allocated and copied from old_isc if old_isc-&gt;gfsw is not NULL</td>
</tr>
</tbody>
</table>

RETURN VALUES
<>NULL Success. The value is a pointer to a new ISC structure.
NULL Failure. The get_new_isc routine was unable to allocate memory for the new ISC structure.

CONSTRAINTS
None
NAME

hw_path_t -- Structure to define a hardware path.

SYNOPSIS

#include <io/gio.h>

DESCRIPTION

The hw_path_t structure defines a hardware path. The first and last indices define a window of address elements that are meaningful. Specify a NULL path by setting the If last_index field to (first_index–1).

STRUCTURE MEMBERS

#define MAX_ELEMENTS  64
#define IO_TREE_LEGACY 0x1

typedef struct hw_path {
  int       flags;
  int       first_index;
  int       last_index;
  hw_addr_t addr[MAX_ELEMENTS];
} hw_path_t;

flags
    If this value is equal to 0, the hardware path of a node is in the new I/O tree view. If the value is equal to IO_TREE_LEGACY, the hardware path of a node is in the legacy I/O tree.

first_index
    An index into the addr array that indicates the first valid element.

last_index
    An index into the addr array that indicates the last valid element.

addr[MAX_ELEMENTS]
    An array of address elements. Only the elements between first_index and last_index are valid.

SEE ALSO

io_hw_path_to_node(9F), io_hw_path_to_str(9F), io_node_to_hw_path(9F).
install_driver(9F)

NAME
install_driver - Install a driver's header structure into the CDIO.

SYNOPSIS
#include <gio.h>

int install_driver(
    drive_info_t *drv_info,
    drv_ops_t *drv_ops
);

PARAMETERS
drv_info Pointer to the driver's drv_info_t structure.
drv_ops Pointer to the driver's drv_ops_t structure.

DESCRIPTION
The install_driver CDIO function installs a driver's header structure outside any specific CDIO, typically for pseudo drivers. WSIO drivers must call wsio_install_driver.
See the HP-UX 11i v3 Driver Development Guide for information.

RETURN VALUES
0 Successful completion.
-1 Error. The major number specified for the driver is already in use. The following message is displayed on the system console and in the error-log file:
   install_driver: Install of driver drv-info->name failed.

CONSTRAINTS
None

SEE ALSO
wsio_install_driver(9F),

337
io_convert_dev_t(9S)

NAME
io_convert_dev_t - Structure to pass in arguments to the driver callback for the
CB_NEW_2_LEGACY_DEV and CB_LEGACY_2_NEW_DEV event operation codes.

SYNOPSIS
#include <iogio.h>

STRUCTURE MEMBERS
struct io_convert_dev {
    int     version;
    dev_t   new_dev;
    int     dev_type;
    size_t  *count;
    dev_t   *legacy_dev;
} io_convert_dev_t;

version Specifies the version of the structure. Set to GIO_VERSION_0.
new_dev For CB_NEW_2_LEGACY_DEV, the dev_t which the legacy dev_t are being
requested must be passed in the user in new_dev.
    For CB_LEGACY_2_NEW_DEV, the dev_t of the given legacy dev_t is returned
in new_dev.

dev_type The dev_t type. It can be D_BLK or D_CHR.
count The driver callback must return the count of the dev_t in legacy_dev. The
caller must pass in a pointer to a size_t as the count argument. For
CB_NEW_2_LEGACY_DEV only.

legacy_dev For CB_NEW_2_LEGACY_DEV, the legacy dev_t for which the new SCSI-3
dev_t is requested must be passed in by the caller in legacy_dev.
    For CB_LEGACY_2_NEW_DEV, the array of legacy dev_t returned by the driver
callback; the array must be allocated by the caller. If memory is not enough to
return the dev_t, it returns GIO_E_INSUFF and the count of the dev_t in
count. The caller must reallocate the memory and call io_new_legacy
dev_t.

DESCRIPTION
The io_convert_dev_t structure is used to pass in arguments to the driver callback for the
CB_NEW_2_LEGACY_DEV and CB_LEGACY_2_NEW_DEV event operation codes.

SEE ALSO
io_dev_info_t(9S)

NAME

io_dev_info_t - Structure for getting the dev_t's of the node in the io_get_devs function.

SYNOPSIS

#include <io/gio.h>

STRUCTURE MEMBERS

struct io_dev_info {
    int version;
    dev_t dev;
    uint64_t options;
    int dev_type;
} io_dev_info_t;

Version Reserved. Must be set to GIO_VERSION_0.
dev The dev_t returned.
options The device-specific options associated with a dev_t. The device specific options are 0, if the device does not have any partitions on it.
dev_type The dev_t type. It can be D_BLK or D_CHR.

DESCRIPTION

The io_dev_info_t structure is used to get the dev_t's of the node in the io_get_devs function.

SEE ALSO

io_get_devs(9F).
NAME

io_dev_to_node - Returns the I/O node associated with a dev_t.

SYNOPSIS

#include <io/gio.h>

void io_dev_to_node(
    dev_t dev,
    int dev_type
);

PARAMETERS

dev The dev_t to be converted.
dev_type The device type, either D_CHR or D_BLK.

DESCRIPTION

The io_dev_to_node routine returns an I/O tree node token given a dev_t by calling the get_node entry point of the associated CDIO.

RETURN VALUES

If successful, returns the I/O tree node corresponding to dev. If the node does not exist, returns NULL.

CONSTRAINTS

None

EXAMPLES

void *node;
hw_path_t node_hwp;

node = io_dev_to_node(dev, dev_type);
if (node == NULL) {
    return NULL;
}
else
    err = io_node_to_hw_path(node, NULL, &node_hwp);

SEE ALSO

io_dev_to_options(9F), io_make_dev_t(9S)
NAME

io_dev_to_options - Return device options in a bitmask format.

SYNOPSIS

#include <io/gio.h>

int io_dev_to_options(
    dev_t dev,
    int dev_type,
    uint64 *options
);

PARAMETERS

dev The dev_t of the device whose options the caller wants to retrieve.
dev_type The dev_t type of the device whose options the caller wants to retrieve (either D_BLK or D_CHR).
options Pointer to the device's options in integer bitmask format.

DESCRIPTION

The io_dev_to_options routine is called to get the device-specific options of the given dev_t and dev_type. The device options are returned in the form of an integer bitmask. For partition dev_t-s, the partition number is returned in the options argument. If the dev_t does not have any options, the service returns 0 in options and GIO_SUCCESS.

RETURN VALUES

IO_SUCCESS Success.
IO_FAILURE Failure.

CONSTRAINTS

None

SEE ALSO

io_dev_to_node(9F)
io_events_t(9S)

NAME

io_events_t - Definitions for the operation codes that can be passed to the driver callback routine.

SYNOPSIS

#include <io/gio.h>

DESCRIPTION

The io_events_t operation codes are as follows:

CB_SCAN_ALL Driver callback is called to perform a SCAN_ALL of the node if the node supports the SCAN_ALL operation code.

CB_CONFIG Driver callback is called to configure the node and call wsio_claim_node to claim the node.

CB_UNCONFIG Driver callback is called to cleanup its internal data structures during an unconfig operation.

CB_DESTROY Driver callback is called when an I/O node is being destroyed.

CB_MK_DEV Driver callback is called to return the dev_t of the given node and device-specific options.

CB_GET_DEVS The driver callback is called with this operation code to get existing dev_t's for an I/O node, including the dev_t's with device-specific options.

CB_LEGACY_2_NEW_DEV Driver callback is called to get the new dev_t of the given legacy dev_t.

CB_NEW_2_LEGACY_DEV Driver callback is called to get the legacy dev_t of the given new dev_t.

CB_DRV_BIND Driver callback is called to configure the SCSI-3 LUN and lunpath nodes.

CB_DRV_UNBIND Unbind a driver from an I/O node. Any persistent data that is required to recognize the device in a subsequent scan (for example, WWID to hw_path mapping) must be left intact.

CB_LUN_2_LUNPATH Get the lunpath associated with a LUN.

CB_LUNPATH_2_LUN Get the LUN associated with a lunpath.

CB_GET_DEV_ATTR Driver callback is called to get the device attributes.

SEE ALSO

escsi_ctlr_node_cb(9F), escsi_tgt_node_cb(9F)
io_get_addr(9F)

NAME

io_get_addr - Get the hardware address of an I/O node.

SYNOPSIS

#include <io/gio.h>

int64_t io_get_addr(
    void *handle
);

PARAMETERS

handle The handle of an I/O node.

DESCRIPTION

The io_get_addr service returns the 64-bit hardware address of an I/O tree node. It can be called for nodes in both the SCSI-2 as well as SCSI-3 view. The hardware address returned is the address of the node relative to its parent.

RETURN VALUES

If successful, the function returns the 64-bit hardware address of the I/O node.

CONSTRAINTS

None

EXAMPLES

hw_addr_t my_addr;

my_addr = io_get_addr(my_node);

SEE ALSO

io_get_name(9F), io_get_type(9F), io_node_to_hw_path(9F)
io_get_devs(9F)

NAME

io_get_devs - Returns the dev_ts of a specified SCSI-3 node.

SYNOPSIS

#include <io/gio.h>

int io_get_devs(
    void *node,
    io_dev_info_t *dev_arr,
    size_t *count
);

PARAMETERS

node    SCSI-3 node handle for which dev_ts are requested.
dev_arr A pointer to an io_dev_info_t structure. The caller allocates memory for dev_arr and passes in the size of the array in count. The caller can initially allocate dev_arr of size IO_MAX_DEVS_IN_IOQ (48). If there is not enough memory in dev_arr to return all the dev_ts of the node, io_get_devs returns GIO_ERROR and returns the count of the dev_ts in the count field. The caller must reallocate memory for dev_arr and call io_get_devs again.

count When called, the caller sets count to the size of dev_arr. On return, io_get_devs sets count to the number of dev_ts in dev_arr.

DESCRIPTION

The io_get_devs routine is called to get all the dev_ts of a SCSI-3 node (including character and block dev_ts). It returns all the dev_ts, the device-specific options and the dev_t type in dev_arr. The dev_type field of dev_arr is updated to denote if the dev_t type is D_CHR or D_BLK.

The caller of io_get_devs must allocate memory for dev_arr and free it after io_get_devs returns. The caller can initially allocate dev_arr of size IO_MAX_DEVS_IN_IOQ (48). The caller also passes a pointer to size_t as the count field with the field set to the number of entries in dev_arr.

The io_get_devs returns the dev_ts in dev_arr and the total number of dev_ts in count. If memory allocated for dev_arr is not enough to return all the dev_ts, io_get_devs returns GIO_E_INSUFF and the number of the dev_ts in count. The caller must then reallocate memory for dev_arr and call io_get_devs again.

RETURN VALUES

GIO_SUCCESS Success.
GIO_ERROR Error.
GIO_E_INSUFF Not enough memory has been allocated to return all the dev_ts.
GIO_E_NOT_SUPPORTED The driver callback returned an error.
GIO_E_INVAL_ARGS Invalid input arguments.

CONSTRAINTS

None

EXAMPLES

io_dev_info_t  dev_info;
size_t          num_devts = 1;
int ret;

dev_info.version = WSIO_VERSION_0;
ret = io_get_devs(my_card_node, &dev_info, &num_devts);

SEE ALSO

io_dev_info_t(9S)
io_get_devsw_len(9F)

NAME

io_get_devsw_len - Get the number of entries in the specified switch table.

SYNOPSIS

#include <io/gio.h>

int io_get_devsw_len(
    devsw_table_t table
);

PARAMETERS

table Indicates which device driver table to reference, cdevsw or bdevsw.

DESCRIPTION

This function returns the number of entries in the specified device switch table.

RETURN VALUES

If successful, the function returns the number of entries in the specified device switch table. If the value for table was not valid, returns –1.

CONSTRAINTS

None

EXAMPLES

/* The following example gets the size of the character
 * switch table.
 */

int cdevsw_size;

cdevsw_size = io_get_devsw_len(CDEVSW_ENTRY);

SEE ALSO

devsw_table_t(9S)
io_get_drv_info(9F)

NAME

io_get_drv_info - Get the driver instance structure for a specified I/O tree node.

SYNOPSIS

#include <io/gio.h>

drv_info_t *io_get_drv_info(
    void *node
);

PARAMETERS

node Pointer to an I/O tree node.

DESCRIPTION

The io_get_drv_info function returns a pointer to a drv_info_t structure for a specified I/O tree node.

RETURN VALUES

If successful, returns a pointer to a drv_info_t structure. On error, returns NULL.

CONSTRAINTS

None

EXAMPLES

/* Given the IO node, get the name and class of the driver that
   * has claimed it.
   */

drv_info_t *drv;

if(drv = io_get_drv_info(handle)) {
    printf("Node driver name[%s] class[%s]\n", drv->name,
            drv->io_class);
}

SEE ALSO

drv_info_t(9S), io_get_flags(9F), io_get_instance(9F), io_get_name(9F), io_get_type(9F)
NAME

io_get_flags - Retrieve the flags values for a specified I/O tree node.

SYNOPSIS

#include <io/gio.h>

int io_get_flags(
    void *node
);

PARAMETERS

node    Pointer to an I/O tree node.

DESCRIPTION

The io_get_flags function returns the flags field for the specified I/O tree node.

RETURN VALUES

If successful, returns the flags field. On error, returns 0.

CONSTRAINTS

None

EXAMPLES

/* The following code checks if the F_EXPLICIT_BINDING flag is set on the I/O node. If so, it then gets the name of the I/O node. */

int flags;
char *node_name;

flags = io_get_flags(node);
if ((flags & F_EXPLICIT_BINDING) != 0) {
    node_name = io_get_name(node);
}

SEE ALSO

io_get_drv_info(9F), io_get_instance(9F), io_get_name(9F), io_get_type(9F)
io_get_instance(9F)

NAME

io_get_instance - Retrieve an instance number for a specified I/O tree node.

SYNOPSIS

#include <io/io.h>

int io_get_instance(
    void *node
);

PARAMETERS

node  Pointer to an I/O tree node.

DESCRIPTION

The io_get_instance WSIO function returns the instance number for the specified I/O tree node.

RETURN VALUES

An instance number or GIO_ERROR, if error.

CONSTRAINTS

None

EXAMPLES

/* This code example first checks to see if there is an instance number and found retrieves the drv_info_t information to get the class name. */

int instance;
    drv_info_t *drv;

    if (instance = io_get_instance(node)) >= 0) {
        if (drv = io_get_drv_info(node)) {
            printf("Node class[\%s] instance[\%d]\n", 
                drv->io_class, instance);
        }
    }

SEE ALSO

io_hw_path_to_node(9F), io_hw_path_to_str(9F), io_node_to_hw_path(9F).
**io_get_mapping_t(9S)**

**NAME**

io_get_mapping_t - Structure to hold LUN and lunpath node handles returned by the driver callback routine

**SYNOPSIS**

```c
#include <io/gio.h>

STRUCTURE MEMBERS

struct io_get_mapping {
    void    **out_buf;
    size_t   *count;
    intptr_t  reserved;
} io_get_mapping_t;
```

- **out_buf** The driver callback returns the LUN or lunpath node handles in this array.
- **count** The number of the node handles returned in `out_buf`.
- **reserved** Reserved for future use.

**DESCRIPTION**

The `io_get_mapping` structure contains the LUN or lunpath node handles for a specified node.

**SEE ALSO**

`io_hw_path_to_node(9F), io_hw_path_to_str(9F), io_node_to_hw_path(9F)`.
NAME

io_get_name - Retrieve a pointer to an I/O tree node name.

SYNOPSIS

#include <io/gio.h>

char *io_get_name(
  void *node
);

PARAMETERS

node Pointer to an I/O tree node.

DESCRIPTION

The io_get_name WSIO function returns a pointer to a character string representing the name
of the given I/O tree node.

NOTE: The node name is the same as its driver's name.

RETURN VALUES

If successful, returns a pointer to a character string representing the name of the given I/O tree
node.

CONSTRAINTS

None

EXAMPLES

char *node_name;

/* Retrieve the name of the I/O node. */
node_name = io_get_name(node);

SEE ALSO

io_get_drv_info(9F), io_get_flags(9F), io_get_instance(9F), io_get_type(9F)
io_get_node_relation(9F)

NAME

io_get_node_relation - Retrieve a relative of a specified node token.

SYNOPSIS

#include <sys/libIO.h>

int io_get_node_relation(
    io_token_t node,
    int relationship,
    uint64_t addr,
    io_token_t *relative
);

PARAMETERS

node  The token of an I/O node in the SCSI-3 view of the I/O tree.
relationship Specifies the type of relative for which to search. Valid values are:
                       IO_REL_CHILD, IO_REL_PARENT, or IO_REL_SIBLING.
addr  The optional address of the relative, if child relative to the parent or sibling
       relative to the node.
relative When called, either NULL or the handle of the last relative found.

NOTE: NULL means the relative pointer is pointing to a NULL. It does not
mean a NULL pointer.

On return, the handle of the next relative. If the end of a subtree is reached,
a NULL is returned and the interface returns IO_SUCCESS.

DESCRIPTION

Call the io_get_node_relation WSIO service to get the relative of the I/O node specified by
node. It works only on nodes in the SCSI-3 view of the tree. The relationship indicates what
kind of relative. Valid values are IO_REL_CHILD, IO_REL_PARENT and IO_REL_SIBLING.
The node token of the relative is returned in the parameter “relative”. If either IO_REL_CHILD or IO_REL_SIBLING is specified, the addr and relative parameters are
used to specify which child or sibling. If the relative parameter is NULL, addr is used to identify
a particular child or sibling. If the relative parameter is not NULL, the addr parameter is ignored.
If relative and node are the same, the first child or sibling is returned; otherwise, the next child or
sibling after the one identified by relative is returned.

RETURN VALUES

IO_SUCCESS  The specified relative was found or the end of a subtree was reached.
IO_ERROR    An error occurred and io_errno is set to one of the following:
               IO_E_DCONF_OPEN  The /dev/config is not open.
               IO_E_DCONF_ACCESS The caller does not have read access permission to
                                  the /etc/ioconfig file.
               IO_E_PARM   Parameter error.
               IO_E_SYSCALL Non-specific system error. See errno for more
                           information.
**CONSTRAINTS**
None

**EXAMPLES**

/* This example gets the first child of an I/O node. */

void *parent, *child;

(void) io_get_node_relation(parent, IO_REL_CHILD, i, &child);

**SEE ALSO**

io_get_drv_info(9F), io_get_instance(9F), io_get_name(9F), io_get_type(9F)
io_get_state(9F)

NAME

io_get_state - Get the state of a specified I/O tree node.

SYNOPSIS

#include <io/gio.h>

int io_get_state(
    void *node
); 

PARAMETERS

node Pointer to an I/O tree node.

DESCRIPTION

The io_get_state WSIO function returns the state field of the specified I/O tree node.

RETURN VALUES

If successful, io_get_state returns the state field of the specified I/O tree node.

CONSTRAINTS

None

EXAMPLES

/* The following example checks to see if an IO node is in the
* UNCLAIMED state.
*/
if(io_get_state(node) == S_UNCLAIMED) {
    printf("node is not claimed\n");
}

SEE ALSO

io_get_drv_info(9F), io_get_instance(9F), io_get_name(9F), io_get_type(9F)
NAME

io_get_type -- Get the type for a specified I/O tree node.

SYNOPSIS

#include <io/gio.h>

int io_get_type(
    void *node
);

PARAMETERS

node Pointer to an I/O tree node.

DESCRIPTION

The io_get_type WSIO function returns the type field of the specified I/O tree node.

RETURN VALUES

If successful, io_get_type returns the type field of the specified I/O tree node.

CONSTRAINTS

None

EXAMPLES

/* The following example checks to see if the IO node type is an interface node.
 */
if(io_get_type(node) == T_INTERFACE) {
    printf("Node is of an interface node\n");
}

SEE ALSO
**io_hw_path_to_node(9F)**

**NAME**

io_hw_path_to_node - Convert a hardware path to an I/O tree node token.

**SYNOPSIS**

```
#include <io/gio.h>

void io_hw_path_to_node(
    void *node,
    hw_path_t *hw_path
);
```

**PARAMETERS**

- `node` Pointer to an I/O node name.
- `hw_path` Pointer to a hardware path.

**DESCRIPTION**

The `io_hw_path_to_node` WSIO function is used to retrieve the I/O tree node at the path specified in `hw_path` relative to `node`. If `node` is NULL, `hw_path` specifies the entire hardware path.

**RETURN VALUES**

The requested node token or NULL, if the specified node does not exist.

**CONSTRAINTS**

None

**SEE ALSO**

`io_hw_path_to_str(9F), io_node_to_hw_path(9F), io_str_to_hw_path(9F)`.
NAME

io_hw_path_to_str - Convert a hardware path to a character string.

SYNOPSIS

#include <io/gio.h>
#include <io/ioparams.h>

int io_hw_path_to_str(
    char *str,
    void *node,
    hw_path_t *hw_path
);

PARAMETERS

str On return, pointer to a hardware path character string.
node Pointer to an I/O node name.
hw_path Pointer to a hardware path.

DESCRIPTION

The io_hw_path_to_str WSIO function converts the path specified by hw_path relative to
node to the string pointed to by str. If node is NULL, hw_path specifies the entire hardware path.
The str parameter must have enough space to hold the converted string. The maximum length
is specified by MAX_HW_PATH_STR. If str is NULL, no data is transferred; the length of the string
is still returned. The string will contain slash (/) separators after bus nexus nodes and period (.)
separators after other node types. Therefore, a typical string might look like 8/4/1.2.0, where
the first two addresses correspond to bus nexus nodes.

If io_hw_path_to_str encounters any aliases on the path from the root node to hw_path, it
substitutes the alias in place of that segment. For example, for hardware path \{1, 34, 24,
1, 2, 4\} and an alias \{1, 34, 24\} equal to aaabbb, the resulting hw_path string is
aaabbb/1.2.4.

RETURN VALUES

GIO_ERROR The specified node does not exist.

CONSTRAINTS

None

EXAMPLES

{
    hw_path_t hw_path;
    char my_string[MAX_HW_PATH_STR];
    io_node_to_hw_path(my_node, NULL, &hw_path);
    io_hw_path_to_str(my_string, NULL, &hw_path);
    printf("%s\n", my_string);
}

SEE ALSO

io_hw_path_to_node(9F), io_node_to_hw_path(9F), io_str_to_hw_path(9F).
**io_init_hw_path(9F)**

**NAME**

io_init_hw_path - Initialize a hardware path structure.

**SYNOPSIS**

```c
#include <io/gio.h>

void io_init_hw_path(
    hw_path_t *hw_path,
    int flags
);
```

**PARAMETERS**

- **hw_path**  
  Pointer to a hardware path structure that is allocated by the caller.

- **flags**  
  The flags field of the hardware path. The only valid flag is IO_TREE_LEGACY, which indicates that the hw_path describes a legacy SCSI-2 node.

**DESCRIPTION**

The io_init_hw_path WSIO service initializes a hardware path structure allocated by the caller. The hardware path is initialized to a null path and the flags field set to the value specified by the caller in the flags parameter. A null path has first_index set to 0 and last_index set to -1 (it does not describe any node in the I/O tree).

**RETURN VALUES**

None

**CONSTRAINTS**

None

**SEE ALSO**

- `hw_path_t(9S)`
io_invoke_devsw(9F)

NAME

io_invoke_devsw - Call driver entry points in the cdevsw or bdevsw table.

SYNOPSIS

#include <io/gio.h>

int io_invoke_devsw(
    devsw_table_t table,
    uint32_t major,
    int *ret,
    devsw_entry_t devsw_entry ...
);

PARAMETERS

- **table**: Indicates the device driver table to reference, either cdevsw or bdevsw.
- **major**: The major number of the device driver that is used to index into the specified switch table.
- **ret**: Pointer to an integer indicating the status of the called driver entry point.
- **devsw_entry**: Indicates the driver entry point to invoke.

DESCRIPTION

The io_invoke_devsw function invokes a driver entry point in either the character or block device driver switch table. Any parameters after `devsw_entry` are passed to the switch table entry point.

RETURN VALUES

- **GIO_SUCCESS**: The driver entry point was called.
- **GIO_ERROR**: The service could not call the driver entry point.

CONSTRAINTS

None

SEE ALSO

devsw_entry_t(9S), devsw_table_t(9S)
io_is_legacy_dev(9F)

NAME

io_is_legacy_dev - Determine if the specified dev_t is a legacy dev_t.

SYNOPSIS

#include <sys/libIO.h>

int io_is_legacy_dev(
    dev_t dev,
    int dev_type
);

PARAMETERS

dev The dev_t on which to determine its style.
dev_type The device type, either D_CHR or D_BLK.

DESCRIPTION

The io_is_legacy_dev WSIO service determines if the specified dev_t is a legacy or non-legacy dev_t. The former includes SCSI-2 dev_t's; the latter includes all others.

RETURN VALUES

0 Not a legacy dev_t.
1 Is a legacy dev_t.

IO_ERROR An error occurred and io_errno is set to one of the following:
  IO_E_DCONF_OPEN The /dev/config is not open.
  IO_E_DCONF_ACCESS The caller does not have read access permission to the /etc/ioconfig file.
  IO_E_PARM Parameter error.
  IO_E_SYSCALL Non-specific system error. See errno for more information.

CONSTRAINTS

None

SEE ALSO

io_is_legacy_node(9F)
io_is_legacy_node(9F)

NAME

io_is_legacy_node - Determine if an I/O node is a legacy (SCSI-2) node.

SYNOPSIS

#include <io/gio.h>

int io_is_legacy_node(
    void *handle
);

PARAMETERS

handle The handle of the I/O node.

DESCRIPTION

The io_is_legacy_node WSIO service returns TRUE if an I/O node is a legacy SCSI-2 node; otherwise, it returns FALSE. Non-legacy nodes include the new-style SCSI-3 nodes and equivalent nodes (for example, CEC and non-mass storage nodes).

RETURN VALUES

FALSE The I/O node is SCSI-3 I/O node or equivalent node.
TRUE The I/O node is a legacy I/O node.
GIO_ERROR Invalid node handle or any other error.

CONSTRAINTS

None

EXAMPLES

/* In the following example the code checks to see if this is a 
* SCSI-2 device node by first checking the node type and then 
* checking if it is a legacy node. 
*/

if(io_get_type(node) == T_DEVICE && io_is_legacy_node(node) {
    printf("I/O node is a SCSI-2 node\n");
}

SEE ALSO

io_get_type(9F)
io_mkdev(9F)

NAME

io_mkdev - Create dev_ts for SCSI-2 nodes or query the dev_t of a SCSI-2 node.

SYNOPSIS

#include <io/gio.h>

dev_t io_mkdev(
    void *node,
    int dev_type,
    char *options
);

PARAMETERS

node Node handle of a SCSI-2 node for which to create a dev_t.

dev_type The device type, either D_BLK or D_CHR.

options Device-specific options for creating SCSI-2 dev_ts in the form of an integer bit mask.

DESCRIPTION

The io_mkdev WSIO service creates dev_ts for SCSI-2 devices or queries the dev_t of a specified SCSI-2 node. It returns the dev_t for a specified SCSI-2 node handle, the device type (D_BLK or D_CHR), and the device-specific options (in the form of an integer bit mask).

RETURN VALUES

If successful, returns the dev_t for a specified SCSI-2 node handle.

On error, returns (dev_t)-1.

CONSTRAINTS

None

SEE ALSO

io_dev_to_options(9F), io_get_devs(9F), io_is_legacy_dev(9F)
io_make_dev_t(9S)

NAME

io_make_dev_t - Structure passed to a driver callback for the CB_MK_DEV event.

SYNOPSIS

#include <io/gio.h>

STRUCTURE MEMBERS

struct io_make_dev {
    int version;
    dev_t dev;
    int dev_type;
    uint64_t options;
} io_make_dev_t;

version Specifies the version of the structure. GIO_VERSION_0 for HP-UX 11i v3.

dev The dev_t returned by the driver callback routine.

dev_type The device type, either D_BLK or D_CHR. It is passed in as input to the driver callback.

options The device-specific options passed in as input to the driver callback. The options field is specified as an integer bit mask.

DESCRIPTION

Structure passed to a driver callback for the CB_MK_DEV event.

SEE ALSO

io_node_callback(9F).
NAME

io_mkdev_ext - Create dev_ts for SCSI-3 nodes or query the dev_t of a SCSI-3 node.

SYNOPSIS

#include <io/gio.h>

#include <io/gio.h>

dev_t io_mkdev_ext(
    void *node,
    int dev_type,
    uint64_t options
);

PARAMETERS

node Node handle of a SCSI-3 node for which to create a dev_t.

dev_type Device type, either type D_BLK or D_CHR.

options Device-specific options for creating new (SCSI-3) dev_ts in the form of an integer bit mask.

DESCRIPTION

The io_mkdev_ext WSIO service creates dev_ts for SCSI-3 devices or queries the dev_t of a specified SCSI-3 node. It returns the dev_t for a specified SCSI-3 node handle, the device type (D_BLK or D_CHR), and the device-specific options (in the form of an integer bit mask).

RETURN VALUES

If successful, returns the dev_t for a specified SCSI-3 node handle. On error, returns NODEV.

CONSTRAINTS

None

SEE ALSO

io_dev_to_options(9F), io_get_devs(9F), io_is_legacy_dev(9F).
io_node_to_hw_path(9F)

NAME

io_node_to_hw_path - Convert an I/O tree node token to a hardware path.

SYNOPSIS

#include <io/gio.h>

int io_node_to_hw_path(
    void *node,
    void *base,
    hw_path_t *hw_path
);

PARAMETERS

node Pointer to an I/O node name.
base Pointer to a base I/O node.
hw_path Pointer to a hardware path.

DESCRIPTION

The io_node_to_hw_path WSIO function fills hw_path with the hardware path relative to base of the given node. If base is NULL, hw_path indicates the full hardware path.

RETURN VALUES

GIO_ERROR The node parameter is not a descendent of base.
GIO_SUCCESS The I/O node token was successfully converted.

CONSTRAINTS

None

SEE ALSO

io_hw_path_to_node(9F), io_hw_path_to_str(9F), io_str_to_hw_path(9F).
NAME

io_post_event - Post an event for an I/O node.

SYNOPSIS

#include <sys/buf.h>

int io_post_event(
    io_post_event_req_t *evt_req
);

PARAMETERS

evt_req Pointer to a io_post_event_req_t structure.

DESCRIPTION

The io_post_event WSIO function is used by drivers

RETURN VALUES

0 Successful completion.
<>0 Error

CONSTRAINTS

Do not call during the boot sequence (during I/O virtual mode configuration); io_post_event is effectively a non-operation.

EXAMPLES

io_post_event_req_t evt_req;
void *io_handle;
io_hw_attr hwattr;
etv_req.flags = IO_EVT_CIM;
etv_req.handle = io_handle;
etv_req.name = "foo";
etv_req.subname = "bar";
etv_req.hw_attr = hwattr;
etv_req.num_named_vars = 0;
etv_req.named_vars_p = NULL;
io_post_event(&evt_req);

SEE ALSO

io_post_event_req_t(9S)
NAME
io_post_event_req_t - Event information posting structure

SYNOPSIS
#include <sys/wsio.h>

DESCRIPTION
The io_post_event_req_t structure WSIO structure type, defined in <sys/wsio.h>, contains event information fields for WSIO drivers. Use this structure to post I/O infrastructure events when calling the service io_post_event.

STRUCTURE MEMBERS
typedef struct io_post_event_req {
    int              version;
    io_evt_flags_t   flags;
    void            *handle;
    EvmPriority_t    priority;
    EvmString_t      name;
    EvmString_t      subname;
    io_hw_attr_t     hw_attr;
    uint32_t         num_named_vars;
    EvmVarStruct_t  *named_vars_p;
} io_post_event_req_t;

version  Data structure version number.

flags  A bit mask that indicates the following event posting options:
    IO_EVT_CIM      The event needs to be converted to a WBEM indication. The event also includes binary data that is a collection of hardware attributes. If you use this flag, do not use EMS->WBEM at the same time to avoid duplication of the WBEM indication.

    IO_EVT_PRIORITY  The structure contains an EVM Event Priority. The priority field overrides the default event priority, which is specified in the event template.

handle  Pointer to the I/O handle for this I/O node.

priority  An event priority to use instead of the default event priority. The default event priority is 200 (Notices event). This field is ignored unless IO_EVT_PRIORITY is set in the flags field.

name  The event name. The event name is appended to the full event name string after the event family name (sys.unix.io). For example, sys.unix.io.name.

subname  The event subname. The event subname is appended to the full event name after the event family name and event name. For example, sys.unix.io.name.subname. This field can be NULL.

hw_attr  An io_hw_attr_t structure. This field is ignored unless IO_EVT_CIM is set in the flags field.

num_named_vars  The number of named variables specified in the named_vars_p array.

named_vars_p  Pointer to an array of type EvmVarStruct_t. The caller uses these fields to add named variables to the EVM event. If the variable exists in the template file, the value is set.
SEE ALSO

io_post_event(9F)
io_query(9F)

NAME

io_query - Get information for a specified token.

SYNOPSIS

#include <io/gio.h>

int io_query(
    io_token_t token,
    int type,
    char *key,
    void *ptr
);

PARAMETERS

token  A value previously obtained by a call to io_search.

type   Specifies one of the following search types:
        S_IOTREE   Search the I/O tree based on the given criteria.
        S_BDEVSW   Search the block device switch table.
        S_CDEVSW   Search the character device switch table.

key    Specifies a string corresponding to a defined query field. For a list of keys, see io_search(9F).

ptr    Pointer to a buffer containing data returned for the corresponding key.

DESCRIPTION

The io_query WSIO routine gets information about a token previously obtained by a call to io_search. The data corresponding to key is copied to the buffer pointed to by ptr. The key parameter must be one of the strings listed in io_search(9F). If ptr is NULL, io_query returns the number of bytes that would have been copied.

RETURN VALUES

Number of bytes copied

0 The key is undefined for this token.

CONSTRAINTS

None

SEE ALSO

io_search(9F)
io_scanall_t(9S)

NAME

io_scanall_t - Structure passed to driver callback for scanning information.

SYNOPSIS

#include <io/gio.h>

STRUCTURE MEMBERS

struct io_scanall {
    hw_path_t hw_path;
    uint64_t flags;
} io_scanall_t;

hw_path  The hardware path of I/O node to be scanned.
flags     Reserved.

DESCRIPTION

Structure passed to driver callback for CB_SCAN_ALL operation code.

SEE ALSO

io_events_t(9S)
NAME

io_search – Search I/O system data structures.

SYNOPSIS

#include <sys/ioparams.h>

void *io_search(
    void *token,
    int type,
    int qual,
    char *key1,
    void *dat1 ...,
    char *keyN,
    void *datN,
    NULL
);

PARAMETERS

token A token returned by a previous call to io_search, or NULL if this is the first search.
type Specifies one of the following search types:
    S_IOTREE Search the I/O tree based on the given criteria.
    S_BDEVSW Search the block device switch table.
    S_CDEVSW Search the character device switch table.

qual Specifies one or more qualifiers for search. Use OR to specify more than one. Most
    qualifiers work for the S_IOTREE searches only. Defined values are:
    Q_SW Software is associated with an I/O tree node.
    Q_HW Hardware is associated with an I/O tree node.
    Q_PSEUDO Is a pseudo device driver. Device switch table searches only.
    Q_DEVSW Has switch table entry points (for example, LDM).
    Q_NEW Newly found hardware.
    Q_TRANS Allows transparent nodes to be returned.
    Q_GIO_INT Allows GIO internal nodes to be returned (deleted nodes).
    Q_CONSOLE Matches the console device.
    Q_DUMP Matches dump devices.
    Q_ROOT Matches root filesystem devices.
    Q_SWAP Matches the swap device.
    Q_BOOT Matches the boot device.
    Q_SAVE_CONF Matches if the DRV_SAVE_CONF flag is set for the driver associated
        with the node.
    Q_DYN_MAJOR Matches if the major can be dynamically allocated. Device switch
        table searches only.
    Q_VIRTBUS Matches software controlled buses (virtual buses) and the virtual
        root node.

key Specifies a string corresponding to a defined query field.


**dat**  Specifies a pointer to data to match for the query field indicated in **key**. There must be a **dat** for every listed **key**.

**DESCRIPTION**

The **io_search** WSIO function searches for I/O system information. The search ends on an entry in the I/O tree or one of the other tables that can be specified in **type**. The entry must pass the qualifier test. And the query of all of its keys equal to all of its dats. You can use the **token** parameter with the **io_query** function to obtain information about the token. The defined keys are listed in the following table:

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b_major</td>
<td>int</td>
<td>Block major number.</td>
</tr>
<tr>
<td>bus_type</td>
<td>char *</td>
<td>The bus type associated with this node. For example, scsi, eisa, and pa.</td>
</tr>
<tr>
<td>c_major</td>
<td>int</td>
<td>Character major number.</td>
</tr>
<tr>
<td>cdio</td>
<td>char *</td>
<td>The name of the associated CDIO. For example, wsio, sio, eisa, pa, and core.</td>
</tr>
<tr>
<td>child</td>
<td>void *</td>
<td>Token corresponding to a child of the specified token.</td>
</tr>
<tr>
<td>class</td>
<td>char *</td>
<td>A character string representing the class of device (for example, disk and tape).</td>
</tr>
<tr>
<td>def_binding</td>
<td></td>
<td>For deferred binding support. It returns TRUE for a node token which has the deferred binding flag set.</td>
</tr>
<tr>
<td>deferred_instance</td>
<td></td>
<td>For deferred binding support. It returns the instance number to be used for next system boot.</td>
</tr>
<tr>
<td>deferred_name</td>
<td></td>
<td>For deferred binding support. It returns the driver name to be used for next system boot.</td>
</tr>
<tr>
<td>description</td>
<td>char *</td>
<td>A character string representing the description of the device.</td>
</tr>
<tr>
<td>driver_name</td>
<td>char *</td>
<td>A character string representing the driver’s name or handle (for example, disc30 and sdisk).</td>
</tr>
<tr>
<td>health</td>
<td></td>
<td>The health of the specified token. See the following table for a list of health string values.</td>
</tr>
<tr>
<td>hw_path</td>
<td>hw_path_t</td>
<td>The hardware path of the specified node.</td>
</tr>
<tr>
<td>id_bytes</td>
<td>void *</td>
<td>The ID bytes of the specified node. This key is context dependent. For PA modules, this is the first 16 bytes of the IODC.</td>
</tr>
<tr>
<td>instance</td>
<td>int</td>
<td>The instance number of the specified node.</td>
</tr>
<tr>
<td>io_flags</td>
<td></td>
<td>Returns the <strong>io_flags</strong> of the given node.</td>
</tr>
<tr>
<td>is_block</td>
<td>int</td>
<td>1, if the driver can be accessed through the block switch table; else, 0.</td>
</tr>
<tr>
<td>is_char</td>
<td>int</td>
<td>1, if the driver can be accessed through the character switch table; else, 0.</td>
</tr>
<tr>
<td>is_pseudo</td>
<td>int</td>
<td>1, if the token corresponds to a pseudo device; else, 0.</td>
</tr>
<tr>
<td>is_remote</td>
<td>int</td>
<td>1, if the token corresponds to a remote device; else, 0.</td>
</tr>
<tr>
<td>is_virtbus</td>
<td>int</td>
<td>1, if the token corresponds to a software virtual bus; else, 0.</td>
</tr>
<tr>
<td>minor_num</td>
<td>int</td>
<td>A device number corresponding to the specified token is returned. If anything other than the default minor number is needed, use <strong>io_mkdev</strong>.</td>
</tr>
</tbody>
</table>
### Key Name | Data Type | Description
--- | --- | ---
name | char * | The name of the specified token. By convention, this is equivalent to the driver name.
parent | void * | Token corresponding to the parent of the specified token.
proc_core | | Returns the processor core information.
proc_socket | | Returns the processor socket information.
proc_thread | | Returns the processor thread information.
registers | unsigned int [ ] | Returns an array of pointers representing the register sets for the specified token. If the module has two register sets, an integer array with two elements is returned. For PA modules, element 0 is the HPA and element 1 is the SPA.
registers64 | unsigned long long [ ] | The 64-bit version of registers.
save_conf | int | 1 if this node is to be saved in the ioconfig file; else, 0.
sibling | void * | Token corresponding to the next sibling of the specified token.
state | char * | A character string corresponding to the state (for example, CLAIMED and UNCLAIMED) of the indicated node.
subtree | hw_path_t | When used in io_query, this query behaves like hw_path. When used in io_search, the search passes if the entry is in the subtree of the dat passed in.
type | char * | A character string corresponding to the specified node’s type. For example, T_DEVICE and T_MEMORY.
visibility | hw_path_t | When used in io_query, this query behaves like hw_path. When used in io_search, the search passes if the entry is visible from the dat passed in. The definition of visibility depends on the device’s subsystem. In general, if interfaces can perform I/O to a device, the device is visible.

The following table lists the health attributes and their values:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>String Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEALTH_ONLINE</td>
<td>online</td>
<td>The I/O Node is online and functional. For example, LUN and lunpath are online and ready to issue I/Os.</td>
</tr>
<tr>
<td>HEALTH_OFFLINE</td>
<td>offline</td>
<td>The I/O Node has gone offline. For example, lunpath has gone offline; HBA Controller has gone link down; tgtpath has gone offline; and all paths to the LUN have gone offline causing the LUN to be inaccessible.</td>
</tr>
<tr>
<td>HEALTH_LIMITED</td>
<td>limited</td>
<td>The I/O Node is operating in a performance-degraded (suboptimal) state. For example, an aggregate group (LUN or an iSCSI session) has one of its links offline. Either one or more lunpaths to a LUN are offline, one of more connections in an iSCSI session are offline, or one or more 32 PHYs in a SAS wide port are offline.</td>
</tr>
<tr>
<td>HEALTH BROKEN</td>
<td>broken</td>
<td>The I/O Node has detected an error condition that needs manual administrator intervention. For example, lunpath has an authentication failure on LUN WWID; tgtpath has an authentication failure on Port WWN; and HBA Controller has detected an unrecoverable PCI Error.</td>
</tr>
<tr>
<td>HEALTH TESTING</td>
<td>testing</td>
<td>The I/O Node is in a diagnostic mode and is being subject to Testing operations. For example, HBA Controller is in a loopback mode; LUN or lunpath is being subject to diagnostic testing.</td>
</tr>
</tbody>
</table>
The IO/Node has been disabled (suspended). For example, PCI OLR Suspend has been performed on the HBA Controller. The HBA Controller has been disabled by an HBA specific command (for example, fcmutil disable). The LUN has been suspended. The LUN has been suspended due to an unrecoverable deferred error. The lunpath has been suspended.

The I/O Node is in a standby state. It is functionally online, but is not in use and is ready to serve as a standby in case of failure of the active in-use I/O Node. For example, alunpath is in a standby state in case of active/passive devices.

**RETURN VALUES**

If successful, io_search returns a token for use with later calls to io_search or io_query. If no match found, the function returns NULL.

**CONSTRAINTS**

None

**EXAMPLES**

The following example searches for all disks in the system and prints the hardware path:

```c
token = NULL;

while (token != NULL) {
    hw_path_t hw_path;
    char hw_path_s[MAX_HW_PATH];
    token = io_search(token,S_IOTREE,Q_HW,"class","disk",NULL);
    if (token != NULL) {
        io_query(token,S_IOTREE,"hw path", &hw_path);
        io_hw_path_to_str(hw_path_s, NULL, &hw_path);
        printf("%s\n", hw_path_s);
    }
}
```

The following example prints the names of all character pseudo drivers in the system:

```c
token = NULL;

while (token != NULL) {
    char pseudo_name[MAX_NAME_LEN];
    token = io_search(token,S_CDEVSW,Q_PSEUDO,NULL);
    if (token != NULL) {
        io_query(token,S_CDEVSW,"driver name", pseudo_name);
        printf("driver name = %s\n", pseudo_name);
    }
}
```

The following example prints the character major number and instance number of all tapes in the system that are SIO-style drivers:

```c
*token = NULL;
key[0] = "class"; ptr[0] = (void *)"tape";
key[1] = "cdio"; ptr[1] = (void *)"SIO";

while (token != NULL) {
    int c_major,instance;
    token = io_search_array(token,S_IOTREE,0,key,ptr);
    if (token != NULL) {
        io_query(token,S_IOTREE,"c major", &c_major);
        io_query(token,S_IOTREE,"instance", &instance);
        printf("%d,%d\n", c_major, instance);
    }
}
```
printf("cmajor=%d instance=%d\n", c_major, instance);

} while (token != NULL);

SEE ALSO

io_query(9F)
io_str_to_class(9F)

NAME

io_str_to_class – Retrieve a class token for specified class name

SYNOPSIS

#include <io/gio.h>
void *io_str_to_class(
    char *name
);

PARAMETERS

name Pointer to a class name for which to retrieve the class token.

DESCRIPTION

The io_str_to_class function returns a class token that can be used as an argument to the class operators. If a class is not defined, GIO creates it.

RETURN VALUES

If successful, io_str_to_class returns a class token. If no class is associated with the name, the function returns NULL.

CONSTRAINTS

None

SEE ALSO

class_get_node(9F).
io_str_to_hw_path(9F)

NAME

io_str_to_hw_path - Convert a character string to a hardware path.

SYNOPSIS

#include <io/gio.h>

int io_str_to_hw_path(
    char *str,
    void *node,
    hw_path_t *hw_path
);

PARAMETERS

str Pointer to a hardware path character string.
node Pointer to an I/O node name.
hw_path Pointer to a hardware path.

DESCRIPTION

The io_str_to_hw_path WSIO function converts a string representing the partial path from node and returns it in hw_path with the hardware path relative to node. If node is NULL, hw_path indicates the full hardware path.

The hw_path parameter receives the full hardware path; however, its first_index points to where the relative hw_path starts.

If str contains an invalid hardware path, io_str_to_hw_path completes as long as it is possible to do so. If str contains a non-existent alias reference or a number is too large to be in one address level, the function fails.

RETURN VALUES

GIO_ERROR The string does not represent a valid path and it is impossible to create one.
GIO_SUCCESS The string was successfully converted.

CONSTRAINTS

None

EXAMPLES

char *str_path="es/456fff780787c56a";
char *str_path2="4/5.4.0";
hw_path_t hw_path;
hw_path_t hw_path_2;

convert_path() {
    io_str_to_hw_path(str_path,NULL,&hw_path);
    io_str_to_hw_path(str_path2,NULL,&hw_path2);
}

SEE ALSO

io_hw_path_to_node(9F), io_hw_path_to_str(9F), io_node_to_hw_path(9F).
iodone(9F)

NAME
io<do`n~ Complete the buffer I/O transaction.

SYNOPSIS
#include <sys/buf.h>
int iodone(
    struct buf *bp
);

PARAMETERS
bp ✽ Pointer to a buf structure.

DESCRIPTION
The iodone WSIO function is used by legacy drivers as an alias for biodone. New drivers must call biodone directly instead of calling iodone.

The biodone kernel function completes the buffer I/O transaction. There must be a corresponding call to biowait for the same bp.

If B_CALL is set in bp->b_flags, biodone calls the callback function specified in bp->b_iodone. The callback function is expected to set the B_DONE flag in bp->b_flags.

If B_CALL is not set in bp->b_flags, biodone marks the buffer I/O as completed by setting the B_DONE flag in bp->b_flags. If B_ASYNC is set, biodone releases the buf structure and associated buffer pointed to by bp, else it resumes the thread waiting on the corresponding call to biowait.

RETURN VALUES
None

CONSTRAINTS
Do not call while holding a spinlock of order >= BUF_HASH_LOCK_ORDER.

WARNINGS
The biodone function calls panic if B_DONE is set in bp->b_flags upon entry.

SEE ALSO
biodone(9F), biowait(9F), iowait(9F)
NAME

iowait - Wait for the buffer I/O to complete.

SYNOPSIS

#include <sys/buf.h>

int iowait(
           struct buf *bp
       );

PARAMETERS

bp   Pointer to a buf structure.

DESCRIPTION

The iowait WSIO function is used by legacy drivers as an alias for biowait. New drivers must
call biowait directly instead of calling iowait.

The biowait kernel function waits for the completion of the buffer I/O specified by bp. A
corresponding call to biodone is required to resume the waiting thread.

RETURN VALUES

0  Successful completion.
<>0  Error

CONSTRAINTS

Do not call in an interrupt context.
Do not call while holding a spinlock.

EXAMPLES

int error;
struct buf *bp;

error = iowait(bp);

/*
 * iowait() returns 0 if the IO completes successfully.
 * A non-zero value is returned if an error has been
 * encountered, however, the error value returned is not
 * always for the IO completion. To get the IO
 * completion error that is returned with the buf, we
 * need to call geterror().
 */
if (error) {
    error = geterror(bp);
}

SEE ALSO

biodone(9F), biowait(9F), geterror(9F), iodone(9F)
isc_claim(9F)

NAME
isc_claim - Marks an ISC entry as claimed by the driver.

SYNOPSIS
#include <sys/io.h>
#include <sys/wsio.h>

void isc_claim(
    struct isc_table_type *isc,
    wsio_drv_info_t drv_info
);

PARAMETERS
isc    Pointer to the ISC entry associated with an interface card or device.
drv_info Pointer to the wsio_drv_info_t structure of the driver that is claiming the ISC structure.

DESCRIPTION
The isc_claim WSIO function marks an ISC entry as claimed by the driver. The isc_claim function is called in the driver_attach function when the driver wants to be assigned to the device represented by the ISC entry.

If drv_info is NULL, the driver is indicating the ISC entry must be discarded. An example of this situation is when the PS2 keyboard driver encounters its second ISC entry. Since the driver only uses the first ISC entry, it can claim and discard the second ISC entry by passing NULL as drv_info.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
static int
mydrv_attach(uint32_t id, struct isc_table_type *isc)
{
    ...
    if (id == MY_DEVICE_HW_ID) {
        /*
         * Specify the interface init function that is
         * called for each claimed ISC entry after the
         * attach chain processing has completed.
         */
        isc->gfsw->init = mydrv_if_init;

        /*
         * Claim the ISC entry representing the device.
         */
        isc_claim(isc, &mydrv_info);
    }

    /*
     * Call the next driver on the attach chain.
     */
    return (*mydrv_saved_attach)(id, isc);
}
SEE ALSO

driver_attach(9E)
NAME
isc_table_type - ISC table entry structure.

SYNOPSIS
#include <sys/io.h>

DESCRIPTION
Each interface card (each device claimed by an interface driver) has an associated Interface Select Code (ISC) entry. The contents of an ISC entry are declared as the isc_table_type structure in <sys/io.h>.

WSIO uses the pointer to an ISC entry as the handle to an interface card. Many WSIO functions require the pointer to an ISC as an argument. WSIO also passes card-specific information to an interface driver through fields in the ISC entry.

Interface drivers use the ISC to store driver-specific information. Some fields in the ISC entry are defined by system and drivers are expected to use these fields as intended by the system. Other fields are available to the driver for its internal use.

I/O Switch Tables
The I/O system supports two I/O interface switch tables through fields in the ISC structure. The isc->gfsw is intended for use by the system. The isc->ifsw is available to specify communication between device and interface drivers.

Generic Function Switch
The generic function switch, isc->gfsw, is intended for system-to-interface driver communication. The table consists of pointers to two function routines:

- isc->gfsw->init — Points to a driver-defined interface initialization routine that is called after all calls to driver_attach functions have been made.
- isc->gfsw->diag — Points to a driver-defined interface diagnostic routine. Its usage is currently not implemented, and drivers must set this pointer to NULL.

Interface Function Switch
The interface function switch, isc->ifsw, is intended for device-to-interface driver communication. It is through this table that a device driver calls its corresponding interface driver. The table is defined and initialized by the interface driver.

For an example of an interface switch table, see the scsi_ifsw structure in <wsio/scsi_ctl.h>.

STRUCTURE MEMBERS
Table 2-2: “Driver Relevant Structure Fields” shows a list of driver accessible fields in the ISC structure that are initialized by the system.

Table 2-2 Driver Relevant Structure Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus_type</td>
<td>int</td>
</tr>
<tr>
<td>if_id</td>
<td>int</td>
</tr>
<tr>
<td>if_info</td>
<td>caddr_t</td>
</tr>
<tr>
<td>if_info-&gt;flags</td>
<td>int</td>
</tr>
<tr>
<td>if_reg_ptr</td>
<td>caddr_t</td>
</tr>
</tbody>
</table>
bus_type

Type of I/O bus for the interface card. For example, PCI_BUS for PCI interface cards.

if_id

Hardware ID of the interface card. The contents of this field are bus_type dependent.

if_info

Pointer to a wsio_if_info structure declared in <sys/wsio.h>.

if_info->flags

Flags indicating the result of an interface driver attempting to claim an interface card. Currently defined values are:

- HAS_IOCHKERR: The card has an I/O check error.
- INITIALIZED: An interface drivers attach routine has successfully initialized the card. This flag is set by the isc_claim function.
- INIT_ERROR: An interface driver’s attach routine tried to claim this card, but failed.
- IS_ISA_CARD: This card is an ISA card.
- NOALLOC_GDD: Reserved
- SLOT_ADDR_VALID: Reserved

if_reg_ptr

Pointer to the base of the interface card’s memory mapped registers. After initial configuration, i.e., when the interface driver’s attach routine is called, if the driver claims the card you can use this field as desired. For example, if the PCI driver memory space is not mapped due to size constraints, call map_mem_to_host and store the returned virtual address from that call in this field.

Table 2-3 “Driver-Initialized ISC Fields” shows a list of fields in the ISC structure that are initialized by drivers.

Table 2-3 Driver-Initialized ISC Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>card_ptr</td>
<td>volatile int *</td>
</tr>
<tr>
<td>gfsw-&gt;init</td>
<td>int (*)(struct isc_table_type *)</td>
</tr>
<tr>
<td>if_drv_data</td>
<td>caddr_t</td>
</tr>
<tr>
<td>if_isc</td>
<td>void *</td>
</tr>
<tr>
<td>ifsw</td>
<td>caddr_t</td>
</tr>
<tr>
<td>my_address</td>
<td>char</td>
</tr>
<tr>
<td>owner</td>
<td>struct buf *</td>
</tr>
<tr>
<td>state</td>
<td>unsigned int</td>
</tr>
</tbody>
</table>

- card_ptr: Pointer to a range of memory mapped interface card registers.
- gfsw->init: Pointer to the init function for the interface driver. The system initializes gfsw to point to a generic function switch table. The interface driver is responsible for updating the table.
- if_drv_data: Pointer to a driver specified object.
- if_isc: Pointer to a driver specified object.
- ifsw: Interface driver entry-point switch. It is set by the interface driver and is intended to be an operational interface between a device driver and its interface driver.
- my_address: Can be used as desired. Usually contains the interface card’s bus address.
| **owner** | Can be used as desired. Usually contains a pointer to the active `buf` or I/O request. |
| **state** | Can be used as desired. Usually contains the device state information. |
NAME

m_wsio_funcnum - Get the number of an interface card function.

SYNOPSIS

#include <sys/wsio.h>

int m_wsio_funcnum(
    dev_t dev,
    wsio_drv_info_t *drv_hdr_ptr
);

PARAMETERS

dev        The dev_t number of a device.
drv_hdr_ptr Pointer to the wsio_drv_info_t structure for the device.

DESCRIPTION

The m_wsio_funcnum WSIO function returns the number of the interface card function associated with device number dev.

RETURN VALUES

None

CONSTRAINTS

None

SEE ALSO
NAME

m_wsio_selcode - Get the select code for a device.

SYNOPSIS

#include <sys/wsio.h>

int m_wsio_selcode(
    dev_t dev,
    wsio_drv_info_t *drv_hdr_ptr
);

PARAMETERS

dev The dev_t number of a device.
drv_hdr_ptr Pointer to the wsio_drv_info_t structure for the device.

DESCRIPTION

The m_wsio_selcode WSIO function returns the select code associated with device number dev.

RETURN VALUES

None

CONSTRAINTS

None

SEE ALSO
NAME

m_wsio_vsc - Return the system bus module number for a device number.

SYNOPSIS

#include <sys/wsio.h>

int m_wsio_vsc(
    dev_t dev,
    wsio_drv_info_t *drv_hdr_ptr
);

PARAMETERS

dev The dev_t number of a device.
drv_hdr_ptr Pointer to the wsio_drv_info_t structure for the device.

DESCRIPTION

The m_wsio_vsc WSIO function returns the system bus module number for device number dev.

RETURN VALUES

None

CONSTRAINTS

None

SEE ALSO
NAME

mod_wsio_attach_list_add - Add the driver attach function pointer to the specified WSIO attach list.

SYNOPSIS

#include <sys/wsio.h>

int mod_wsio_attach_list_add(
    int type,
    void *attach_func,
    char *driver_name
);

PARAMETERS

type Type of WSIO attach list.
attach_func Pointer to the driver attach function.
driver_name Pointer to the driver name.

DESCRIPTION

The mod_wsio_attach_list_add WSIO function adds the driver attach function pointer attach_func to the WSIO attach list specified by type.

Dynamically loadable drivers call mod_wsio_attach_list_add when they are loaded into the kernel.

The type parameter specifies the attach list to use. Valid values are:

MOD_WSIO_CORE For Core I/O attach list
MOD_WSIO_EISA For EISA I/O attach list
MOD_WSIO_PCI For PCI I/O attach list

The attach_func parameter points to the driver attach function that will be called by the WSIO environment to see if the driver wants to claim a device.

RETURN VALUES

0 Successful completion.
1 Error

CONSTRAINTS

None

EXAMPLE

/*
 * Add my driver attach function to the WSIO attach list
 * for claiming PCI devices.
 */
if (mod_wsio_attach_list_add(MOD_WSIO_PCI,
        &mydrv_attach, driver_name)) {
    return ENXIO;    /* attach add failed! */
}

SEE ALSO

mod_wsio_attach_list_remove(9F)
NAME

mod_wsio_attach_list_remove - Remove the driver attach function pointer to the specified WSIO attach list.

SYNOPSIS

#include <sys/wsio.h>

int mod_wsio_attach_list_remove(
    int type,
    void *attach_func
);

PARAMETERS

type Type of WSIO attach list.
attach_func Pointer to the driver attach function.

DESCRIPTION

The mod_wsio_attach_list_remove WSIO function removes the driver attach function pointer attach_func from the WSIO attach list specified by type. The attach_func and type parameters must match the parameters passed to mod_wsio_attach_list_add.

Dynamically loadable drivers call mod_wsio_attach_list_remove when they are unloaded from the kernel.

The type parameter specifies the attach list to use. Valid values are:

MOD_WSIO_CORE For Core I/O attach list
MOD_WSIO_EISA For EISA I/O attach list
MOD_WSIO_PCI For PCI I/O attach list

The attach_func parameter points to the driver attach function.

RETURN VALUES

0 Successful completion.
1 Error
ENOMATCH The attach_func parameter does not match the value passed to mod_wsio_attach_list_add.

CONSTRAINTS

None

EXAMPLE

/*
 * Remove my driver attach function from the WSIO attach list
 * for claiming PCI devices.
 */
if (mod_wsio_attach_list_remove(MOD_WSIO_PCI,
    &mydrv_attach)) {
    return ENXIO; /* attach remove failed! */
}

SEE ALSO

mod_wsio_attach_list_add(9F)
NAME

modlink - Structure contains information about the operations supported by the driver and information about the driver typespecific data.

SYNOPSIS

#include <sys/moddefs.h>

DESCRIPTION

A pointer to the modlink structure is stored in the modwrapper structure. The modlink structure gives information about the operations supported by the driver and the driver typespecific information. The exact contents are dependent on the type of the driver.

STRUCTURE MEMBERS

The modlink structure is defined as follows:

struct modlink {
    struct  mod_operations  *ml_ops;
    void    *ml_type_data;
};

The elements of the modlink structure are shown in Table 2-4 “Modlink Structure”.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml_ops</td>
<td>A pointer to a mod_operations switch table for the module type. The structure depends on the type of DLKM driver module being defined.</td>
</tr>
<tr>
<td>ml_type_data</td>
<td>Pointer to the mod_type_data structure. The mod_type_data contains a module description string and the second field is set to NULL.</td>
</tr>
</tbody>
</table>

EXAMPLE

The name of the modlink structure array must be driver_mod_link. The array of the modlink structure must be terminated by a {NULL, NULL} entry.

The following is an example of a modlink structure array:

static struct mod_type_data driver_drv_link = {
    "Driver description string",
    (void *)NULL
};

struct modlink driver_mod_link[] = {
    &gio_mod_ops, &driver_drv_link, 
    {NULL, NULL}
};

SEE ALSO

driver_load(9E), driver_unload(9E), modwrapper(9S)
modwrapper(9S)

NAME

modwrapper - Structure to specify DLKM driver entry points and configuration information.

SYNOPSIS

#include <sys/moddefs.h>

DESCRIPTION

The modwrapper structure must be named as drivername_wrapper, and must be externally visible.

STRUCTURE MEMBERS

The modwrapper structure is defined as follows:

```c
extern struct mod_conf_data drivername_conf_data;

struct modwrapper {
    int      mw_rev;
    int      (*mw_load) (void *);
    int      (*mw_unload) (void *);
    void     (*mw_halt) (void);
    void     *mw_conf_data;
    struct modlink *mw_modlink;
};
```

The fields of the modwrapper structure are shown in Table 2-5 "Modwrapper Structure".

Table 2-5 Modwrapper Structure

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mw_rev</td>
<td>Driver revision number. The field must always be set to MODREV. MODREV is defined in &lt;sys/moddefs.h&gt;.</td>
</tr>
<tr>
<td>mw_load</td>
<td>Pointer to the driver's load function.</td>
</tr>
<tr>
<td>mw_unload</td>
<td>Pointer to the driver's unload function.</td>
</tr>
<tr>
<td>mw_halt</td>
<td>Reserved for future use. Must be set to NULL.</td>
</tr>
<tr>
<td>mw_conf_data</td>
<td>Pointer to driver's configuration data. This field must always be set to drivername_conf_data. The data structure is automatically generated as part of the driver's metadata.</td>
</tr>
<tr>
<td>mw_modlink</td>
<td>Pointer to an array of structmodlink.</td>
</tr>
</tbody>
</table>

NOTE

The source file that defines the modwrapper structure will need to have an extern for mod_conf_data as drivername_conf_data.

SEE ALSO

driver_load(9E), driver_unload(9E), modlink(9S)
prop_create(9F)

NAME

prop_create - Create a property associated with the given I/O tree node.

SYNOPSIS

#include <io/gio.h>

int prop_create(
    void *node,
    int flags,
    char *name,
    void *val,
    int length
);

PARAMETERS

node Pointer to an I/O tree node token.
flags Flag modifiers.
name A NULL-terminated string containing the name of the property.
val Pointer to the data to associate with the property.
length The number of bytes pointed to by val.

DESCRIPTION

The prop_create WSIO function creates the property name with the data pointed to by val. If length is 0, the function creates a boolean property with no data (val can be NULL in this case).

The flags parameter specifies the modifiers to use. Valid values are:

P_NO_SLEEP Do not sleep while waiting for resources.
P_NO_COPY Do not allocate space for the data; point to the given val. The caller must guarantee that the data never changes.
P_NO_EXPORT Do not export the property to children of the node.
P_MODIFY Modify the property if it already exists.

RETURN VALUES

PROP_ERROR Node does not exist or property already exists, if P_MODIFY is not set.
PROP_NO_MEM Memory allocation failure, if P_NO_SLEEP is set.
PROP_SUCCESS Successful completion.

CONSTRAINTS

None

SEE ALSO

prop_destroy(9F), prop_destroy_all(9F), prop_get(9F), prop_modify(9F), prop_size(9F)
prop_destroy(9F)

NAME

prop_destroy - Remove a property associated with the given I/O tree node.

SYNOPSIS

#include <io/gio.h>

int prop_destroy(
    void *node,
    char *name
);

PARAMETERS

node Pointer to an I/O tree node token.
name A NULL-terminated string containing the name of the property.

DESCRIPTION

The prop_destroy WSIO function removes the property name from the property list for the specified node. The function does not search parent nodes for the property.

RETURN VALUES

PROP_ERROR Invalid arguments. For example, node does not exist.
PROP_NOT_FOUND The named property does not exist for this node.
PROP_SUCCESS Successful completion.

CONSTRAINTS

None

SEE ALSO

prop_create(9F), prop_destroy_all(9F), prop_get(9F), prop_modify(9F), prop_size(9F)
prop_destroy_all(9F)

NAME

prop_destroy_all - Remove all properties associated with the given I/O tree node.

SYNOPSIS

#include <io/gio.h>

int prop_destroy_all(
    void *node
);

PARAMETERS

node Pointer to an I/O tree node token.

DESCRIPTION

The prop_destroy_all WSIO function removes all properties from the property list for the specified node. Properties inherited from parent nodes are not affected.

RETURN VALUES

PROP_ERROR Invalid arguments. For example, node does not exist.
PROP_SUCCESS Successful completion.

CONSTRAINTS

None

SEE ALSO

prop_create(9F), prop_destroy(9F), prop_get(9F), prop_modify(9F), prop_size(9F)
NAME

prop_get - Retrieve the value associated with the specified property.

SYNOPSIS

#include <io/gio.h>

int prop_get(
    void *node,
    int flags,
    char *name,
    void *ptr
);

PARAMETERS

node Pointer to an I/O tree node token.
flags Flag modifiers.
name A NULL-terminated string containing the name of the property.
ptr If P_NO_COPY is set, ptr is a pointer to a pointer. Otherwise, ptr points to a buffer into which the data is copied.

DESCRIPTION

The prop_get WSIO function provides access to property data. If P_NO_COPY is set, a pointer to the property data is copied into the location pointed to by ptr. Otherwise, the property data is copied to ptr. If P_NO_SEARCH is set, only properties associated with node are searched. Otherwise, the search continues up the I/O tree until either the property is found or the root node is encountered.

The flags parameter specifies the modifiers to use. Valid values are:

P_NO_COPY Causes a pointer to the property data to be copied to the location pointed to by ptr; the data itself is not copied. Use this pointer only to read data.
P_NO_SEARCH Do not search parent nodes for the definition of the property.

The prop_get routine assumes that the buffer pointed to by ptr is large enough to hold the property data. Use prop_size to determine how much space to allocate.

For boolean properties, the condition is TRUE if the property exists and has a length of 0, or exists, has a length of non-zero, and its value is not equal to 0. It is FALSE otherwise (does not exist, or exists and has a value of 0).

int is_prop(void *node, char *name){
    size_t size;
    size_t val;
    size=prop_size(node, 0, name);
    if (size<0 || size>sizeof(size_t))
        return FALSE;
    if (size==0)
        return TRUE;
    prop_get(node, 0, name, &val);
    return val?TRUE:FALSE;
}

This is preferable to checking for existence of the property alone because in any point in the property’s subtree (where the query of the property is TRUE), one can create a false property and set the query of the property to FALSE only for that deeper subtree.
RETURN VALUES

- PROP_ERROR: Invalid arguments. For example, node does not exist.
- PROP_NOT_FOUND: No property named *name* was found.
- PROP_SUCCESS: Successful completion.

CONSTRAINTS

None

EXAMPLE

Read the IODC bytes for a PA module:

```c
iodc_t iodc;

prop_get(node, P_NO_SEARCH, "id_bytes", &iodc);
```

Get a pointer to the IODC bytes for a PA module:

```c
iodc_t *iodc;

prop_get(node, P_NO_COPY | P_NO_SEARCH, "id_bytes", &iodc);
```

SEE ALSO

- `prop_create(9F)`, `prop_destroy(9F)`, `prop_destroy_all(9F)`, `prop_modify(9F)`, `prop_size(9F)`
prop_modify(9F)

NAME

prop_modify - Modify the value associated with the specified property.

SYNOPSIS

#include <io/gio.h>

int prop_modify(
    void *node,
    int flags,
    char *name,
    void *val,
    int length
) ;

PARAMETERS

node    Pointer to an I/O tree node token.
flags   Flag modifiers.
name    A NULL-terminated string containing the name of the property.
val     Pointer to the data to associate with the property.
length  The number of bytes pointed to by val.

DESCRIPTION

The prop_modify WSIO function modifies the value of the property name. If the length of the property changes as a result of the modification, there might be memory allocation and deallocation necessary. The prop_modify routine is equivalent to a call to prop_destroy followed by a call to prop_create.

The flags parameter specifies the modifiers to use. Valid values are:

P_NO_COPY    Do not allocate space for the data; point to the specified val. The caller must guarantee that the data never changes.

P_NO_SLEEP   Do not sleep waiting for resources.

RETURN VALUES

PROP_ERROR   Invalid arguments. For example, node does not exist.
PROP_NO_MEM  Memory allocation failure, if P_NO_SLEEP is set.
PROP_NOT_FOUND No property name was found.
PROP_SUCCESS Successful completion.

CONSTRAINTS

None

SEE ALSO

prop_create(9F), prop_destroy(9F), prop_destroy_all(9F), prop_get(9F), prop_size(9F)
prop_size(9F)

NAME

prop_size - Determine the size of a specified property.

SYNOPSIS

#include <io/gio.h>

int prop_size(
    void *node,
    int flags,
    char *name
);

PARAMETERS

node   Pointer to an I/O tree node token.
flags  Flag modifiers.
name   A NULL-terminated string containing the name of the property.

DESCRIPTION

The prop_size WSIO function returns the size of the property name, in bytes. If P_NO_SEARCH is set, the function searches only properties associated with node. Otherwise, the search continues up the I/O tree until either the property is found or the root node is encountered.

The flags parameter specifies the modifiers to use. Valid values are:

P_NO_SEARCH   Do not search parent nodes for definition of the property.

RETURN VALUES

If successful, prop_size returns the size of the property, in bytes. If unsuccessful, the function returns one of the following:

PROP_ERROR    Parameter error.
PROP_NOT_FOUND No property name was found.

CONSTRAINTS

None

SEE ALSO

prop_create(9F), prop_destroy(9F), prop_destroy_all(9F), prop_get(9F), prop_modify(9F)
NAME

raw_to_block - Return the corresponding block dev_t given character dev_t.

SYNOPSIS

#include <io/gio.h>

dev_t raw_to_block(
    dev_t dev
);

PARAMETERS

dev Character dev_t.

DESCRIPTION

The raw_to_block function returns the corresponding block dev_t for a character dev_t. It also checks that the passed dev_t is valid.

RETURN VALUES

On success, returns the block dev_t for the passed character dev_t. On error, returns the following:

NODEV The passed dev_t is not valid, or the drv_info for the dev_t is NULL.

CONSTRAINTS

None

SEE ALSO

block_to_raw(9F)
NAME  
wsio_activate_probe - Activate the probe function for a driver.

SYNOPSIS  
void wsio_activate_probe(
    char *probe_name,
    struct drv_info *drv_infop
);

PARAMETERS  
probe_name Name of the device probe function as registered by 
            wsio_register_dev_probe

drv_infop Pointer to the driver drv_info structure

DESCRIPTION  
The wsio_activate_probe WSIO function connects the probe function for a dynamically 
loadable interface driver to the driver drv_info structure. The wsio_activate_probe function 
is called in the driver's load entry point after its device probe function has been registered with 
the WSIO CDIO.

RETURN VALUES  
None

CONSTRAINTS  
None

EXAMPLES  
static wsio_drv_info_t mydrv_info = { ... };  

int mydrv_load(void * arg)  
{
    /*
    * Use the drv_info passed to to the driver as arg
    * instead of using the static version.
    */
    mydrv_info.drv_info = (drv_info_t *)arg;

    /*
    * Register the driver with WSIO.
    * Note: returns 0 on failure.
    */
    if (!wsio_install_driver(&mydrv_info)) {  
        return ENXIO;  /* Install driver failed! */
    }

    /*
    * Add my driver attach function to the WSIO attach list
    * for claiming PCI devices.
    */
    if (mod_wsio_attach_list_add(MOD_WSIO_PCI,
        &mydrv_attach, driver_name)) {
        /*
        * Attach list add failed! Uninstall the driver
        * and return.
        */
        (void)wsio_uninstall_driver(&mydrv_info);  
}
return ENXIO;
}

/*
 * Register the device probe function for the driver.
 */
if (wsio_register_dev_probe(IF_CLASS, mydrv_probe_func, "mydrv_probe")) {
    /*
     * Register device probe failed! Remove driver from
     * the attach list, uninstall the driver and return.
     */
    (void)mod_wsio_attach_list_remove(MOD_WSIO_PCI, &mydrv_attach);
    (void)wsio_uninstall_driver(&mydrv_info);
    return ENXIO;
}

/*
 * The following step is only required for dynamically
 * loadable drivers: connect the probe function.
 */
wsio_activate_probe("mydrv_probe", mydrv_info.drv_info);
return 0;

SEE ALSO

wsio_register_dev_probe(9F), wsio_unregister_dev_probe(9F)
NAME

wsio_alloc_mem - Allocate memory for DMA buffers or control structures.

SYNOPSIS

#include <sys/wsio.h>

wsio_alloc_status_t wsio_alloc_mem(
    wsio_mem_handle_t mem_handle,
    size_t size,
    wsio_vaddr_t *vaddr,
    wsio_alloc_flags_t flags
);

PARAMETERS

mem_handle Handle allocated by a call to wsio_alloc_mem_handle.
size Size of the buffer in bytes.
vaddr Address of the pointer to the allocated buffer. Pointer is set to NULL if unable to allocate a buffer.
flags Flags which describe restrictions:
    WSIO_SLEEP_OK Flag to indicate service can sleep if waiting for resources.

DESCRIPTION

This WSIO service allocates memory used for DMA buffers or control structures. The first parameter to the service must be a mem_handle allocated by a call to wsio_alloc_mem_handle.

RETURN VALUES

WSIO_ALLOC_OK The buffer was allocated.
WSIO_ALLOC_OUT_OF_RESOURCES Unable to allocate the specified resources.

CONSTRAINTS

If WSIO_SLEEP_OK is set in flags:
• Do not call while holding a spinlock
• Do not call in interrupt context.

EXAMPLES

wsio_alloc_status_t status;
wsio_vaddr_t vaddr;
wsio_mem_handle_t mem_handle;
sizet size;

if(wsio_alloc_mem_handle(my_isc, &mem_handle,
    WSIO_32BIT_MEMORY |
    WSIO_IO_CONTIGUOUS) == WSIO_ALLOC_OK)
{
    /* Allocate memory with a non-blocking call to wsio_alloc_mem() */
    status = wsio_alloc_mem(mem_handle, size, &vaddr, 0);
}

SEE ALSO

wsio_alloc_mem_handle(9F), wsio_free_mem(9F), wsio_free_mem_handle(9F)
wsio_alloc_mem_handle(9F)

NAME

wsio_alloc_mem_handle - Specify the type of memory to allocate.

SYNOPSIS

#include <sys/wsio.h>

wsio_alloc_status_t wsio_alloc_mem_handle(
    struct isc_table_type *isc,
    wsio_mem_handle_t *mem_handle,
    wsio_mem_alloc_attrib_t attribs
);

PARAMETERS

isc Pointer to the device isc structure.
mem_handle Pointer to the returned handle.
attribs Attributes describing the criteria for the type of memory to allocate.

WSIO_OPTIMIZE_FOR_DEVICE Allocate memory close to the device. On Multicell systems, memory is allocated on the same cell as the device.

WSIO_OPTIMIZE_FOR_CPU Allocate memory close to the current CPU. On Multicell systems, memory is allocated on the same cell as the CPU. This is the default behavior.

WSIO_32BIT_MEMORY Allocate memory below 4GB.

WSIO_IO_CONTIGUOUS On platforms without an I/O PDIR, physically contiguous memory is allocated.

WSIO_ALIGN_ON_SIZE Allocate memory aligned to a power-of-two value greater than or equal to the requested size.

DESCRIPTION

Drivers call this service to specify the type of memory to allocate. The service returns a mem_handle, which is passed to the WSIO memory alloc and free routines. Drivers can allocate more than one mem_handle to specify different criteria for memory allocation.

While it is not a requirement, HP recommends that drivers call this routine early in their initialization sequence. This is due to the high overhead of the routine. Buffer alignment is as follows:

<table>
<thead>
<tr>
<th>Allocation Size</th>
<th>Buffer Aligned On</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less Than Cacheline Size</td>
<td>32 Byte Boundary</td>
</tr>
<tr>
<td>Greater Than or Equal to Cacheline Size</td>
<td>Cacheline Boundary</td>
</tr>
<tr>
<td>Greater Than or Equal to I/O Page Size (4K)</td>
<td>4K Boundary</td>
</tr>
</tbody>
</table>

RETURN VALUES

WSIO_ALLOC_OK The handle was allocated.
WSIO_ALLOC_OUT_OF_RESOURCES Unable to allocate the specified resources.
WSIO_INVAL_PARAM A parameter was not valid.
CONSTRAINTS
Do not call in interrupt context.
Do not call while holding a spinlock.

EXAMPLE

wsio_alloc_status_t status;
wsio_vaddr_t vaddr;
wsio_mem_handle_t mem_handle;
size_t size;

if(wsio_alloc_mem_handle(my_isc, &mem_handle,
    WSIO_32BIT_MEMORY |
    WSIO_IO_CONTIGUOUS) == WSIO_ALLOC_OK)
{
    /* Allocate memory with a non-blocking call
       to wsio_alloc_mem() */
    status = wsio_alloc_mem(mem_handle, size, &vaddr, 0);
}

SEE ALSO

wsio_alloc_mem(9F), wsio_free_mem(9F), wsio_free_mem_handle(9F)
NAME
wsio_allocate_dma_handle - Obtain a handle used to setup DMA.

SYNOPSIS
#include <sys/wsio.h>
void *wsio_allocate_dma_handle(
    struct isc_table_type *isc
);

PARAMETERS
isc Pointer to the driver's isc_table entry.

DESCRIPTION
The wsio_allocate_dma_handle WSIO function is called by device drivers to obtain a DMA handle. This handle, which is passed to all DMA services, can be associated with various DMA hints, and is used to control DMA. Multiple handles can be allocated, allowing a device driver to associate different hints with each handle.

RETURN VALUES
A void pointer to the handle. If NULL is returned, a handle could not be allocated.

CONSTRAINTS
None

EXAMPLE
void *dma_handle;

dma_handle = wsio_allocate_dma_handle(isc_entry);
if (dma_handle == NULL) {
    /* No handle allocated. */
    return (ERROR);
}
/* The DMA handle is now in the dma_handle variable */

SEE ALSO
wsio_allocate_shared_mem(9F), wsio_dma_pass_thru(9F), wsio_fastmap_dma_buffer(9F),
wsio_free_dma_handle(9F), wsio_free_shared_mem(9F), wsio_flush_shared_mem(9F),
wsio_init_map_context(9F), wsio_iova_to_phys(9F), wsio_map_dma_buffer(9F),
wso_remap_dma_buffer(9F), wsio_set_device_attributes(9F), wsio_set_dma_attributes(9F),
wso_unmap_dma_buffer(9F)
NAME

wsio_allocate_shared_mem - Set up an I/O virtually contiguous DMA buffer.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_allocate_shared_mem(
    struct isc_table_type *isc,
    void *dma_handle,
    size_t size,
    wsio_iova_t *iova,
    wsio_vaddr_t *vaddr,
    wsio_shared_mem_attr_t shared_mem_attr
);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
dma_handle DMA handle allocated using wsio_allocate_dma_handle.
size Size of buffer to allocate.
iova Pointer that contains the I/O virtual address upon completion. A
    wsio_iova_t variable must be allocated by the driver, and the pointer
to this is what must be passed into the macro.
vaddr Pointer that contains the host virtual address upon completion. A
    wsio_vaddr_t variable must be allocated by the driver, and the pointer
to this is what must be passed into the macro.
shared_mem_attr Bitmask that indicates how to allocate the memory. The acceptable values
    are described in the following list. If a type of 0 is used, the default
    behavior of WSIO_IO_SHMEM_OPTIMIZE_DEVICE_LATENCY is used.
    The behavior of the allocation is also affected by attributes set using
    wsio_dma_set_device_attributes, and
    wsio_set_dma_attributes.

The following are the wsio_shared_mem_attr_t allowable bitmask values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_IO_SHMEM_OPTIMIZE_DEVICE_LATENCY</td>
<td>Allocation must optimize for device access latency. If possible, allocate object in memory local to a bus bridge.</td>
</tr>
<tr>
<td>WSIO_IO_SHMEM_OPTIMIZE_HOST_LATENCY</td>
<td>Allocation must optimize for host access latency. If possible, allocate in host memory.</td>
</tr>
<tr>
<td>WSIO_IO_SHMEM_DMA_ALLOC_COMPATIBLE</td>
<td>Behave exactly as the 10.X dma_alloc service. This type is for compatibility with 10.X only.</td>
</tr>
<tr>
<td>WSIO_IO_SHMEM_INBOUND</td>
<td>This attribute can be ORed with the other attributes to indicate the buffer is used exclusively for inbound DMA.</td>
</tr>
<tr>
<td>WSIO_IO_SHMEM_OUTBOUND</td>
<td>This attribute can be ORed with the other attributes to indicate the buffer is used exclusively for outbound DMA.</td>
</tr>
</tbody>
</table>
**DESCRIPTION**

The `wsio_allocate_shared_mem` WSIO function is called by a device driver to allocate an I/O virtually contiguous DMA buffer that is to be used for continuous DMA. Continuous DMA means that the memory appears contiguous to the I/O device and can be read or written by the I/O device on a continuous basis. For packet DMA or DMA that is used for temporary mappings, `wsio_map_dma_buffer` must be used.

If a callback function is set up (see `wsio_set_dma_callback`), and no resources are available when the call is made, `WSIO_MAP_W_CALLBACK` will be returned, and the callback will be triggered when resources become available.

This macro may be called in a non-blocking context.

**RETURN VALUES**

- **WSIO_MAP_OK**: Success
- **WSIO_MAP_W_CALLBACK**: Returned if no resources are available and a callback is registered.
- **WSIO_MAP_E_NO_RESOURCES**: Returned if no resources are available and no callback is registered.
- **WSIO_MAP_ERESOURCE_ERROR**: Returned if cannot allocate resources. If this is returned, the allocation will never succeed.
- **WSIO_MAP_EPARAMETER_ERROR**: Returned on bad parameter (Software bug).
- **WSIO_MAP_EUNKNOWN_ERROR**: Returned if there is an unknown error.

**CONSTRAINTS**

None

**EXAMPLE**

```c
void *dma_handle = NULL;
wsio_iova_t io_virtual_addr;
wsio_vaddr_t host_virtual_addr;

dma_handle = wsio_allocate_dma_handle(isc_entry);
if (dma_handle == NULL) {
    /* No handle allocated. */
    return ERROR;
}

/* The DMA handle is now in the dma_handle variable */

if (wsio_allocate_shared_mem(isc_entry, dma_handle, buf_size, &io_virtual_addr, &host_virtual_addr, 0) != WSIO_MAP_OK) {
    /* Unable to allocate shared memory, so return an error */
    return ERROR;
}

/* A buffer of size 'buf_size' is now allocated and mapped */
```
* in both host virtual memory and I/O virtual memory space.
*/

**SEE ALSO**

`wsio_allocate_dma_handle(9F)`, `wsio_dma_pass_thru(9F)`, `wsio_free_dma_handle(9F)`, `wsio_free_shared_mem(9F)`, `wsio_flush_shared_mem(9F)`, `wsio_iova_to_phys(9F)`, `wsio_set_device_attributes(9F)`, `wsio_set_dma_attributes(9F)`
NAME

wsio_async_scan - Initiate an asynchronous scan from an I/O node.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_async_scan(
    void *handle,
    wsio_async_flags_t flags
);

PARAMETERS

handle Pointer to the handle of the I/O node under which to scan.

flags Scan flags that indicate whether to notify the caller when the scan completes. Reserved for future use. Currently, the only value defined is WSIO_FLAGS_NONE.

DESCRIPTION

The wsio_async_scan WSIO service is called to initiate an asynchronous I/O scan beneath an I/O node.

RETURN VALUES

WSIO_OK The operation was successful.

WSIO_INVALID_HNDL The component handle is invalid.

WSIO_ERROR Failure in memory allocation or thread allocation for scheduling the I/O scan.

CONSTRAINTS

None

SEE ALSO

wsio_claim_node(9F), wsio_destroy_legacy(9F), wsio_get_drv_priv(9F), wsio_put_drv_priv(9F),
wsio_node_get_isc(9F), wsio_reg_legacy(9F), wsio_reg_node(9F), wsio_reg_node_t(9S)
WSIO_BIG_ENDIAN(9F)

NAME

WSIO_BIG_ENDIAN – Function to return true (1) if the local bus is big-endian.

SYNOPSIS

#include <sys/wsio.h>

int WSIO_BIG_ENDIAN(
    struct isc_table_type *isc
);

PARAMETERS

isc Pointer to the driver's isc_table entry.

DESCRIPTION

The WSIO_BIG_ENDIAN macro is called by a device driver to report whether the local bus is big-endian. If it is, true is returned, otherwise it returns false. This can be used by a driver along with the known endianness of the host processor to decide whether endian swapping must be performed. Endian swapping might be necessary for any data transfers between the I/O bus and local host memory.

The WSIO_BIG_ENDIAN macro can be called in a non-blocking context.

RETURN VALUES

1 Local bus is big-endian.
0 Not big-endian.

CONSTRAINTS

None

EXAMPLE

if (WSIO_BIG_ENDIAN(isc_entry)) {
    /* No endian swapping necessary */
} else {
    /* Endian swapping must be performed */
}

SEE ALSO

WSIO_LITTLE_ENDIAN(9F)
**NAME**

wsio_cfg_inXX - Macros to read from configuration space.

**SYNOPSIS**

```c
#include <sys/wsio.h>

void wsio_cfg_inXX(
    struct isc_table_type *isc,
    wsio_addr_handle_t cfg_handle,
    uint32_t offset,
    uintXX_t *data
);
```

**PARAMETERS**

- **isc**
  Pointer to the driver's `isc_table` entry.
- **cfg_handle**
  Configuration space handle.
- **offset**
  Byte offset into the configuration space.
- **data**
  Pointer to an appropriately sized and aligned memory space for the returned data.

**DESCRIPTION**

The `wsio_cfg_inXX` macros are called by device drivers to read from configuration space. The `cfg_handle` and the `offset` are used to specify the correct location to read from. The value `XX` refers to 8, 16, 32, or 64, and indicates the amount of data to read from configuration space. Endian translation is performed automatically if the host memory and local bus have different endianness.

**RETURN VALUES**

None

**CONSTRAINTS**

None

**EXAMPLE**

```c
wsio_addr_handle_t handle;
uint32_t data;

if (wsio_map_cfg_handle(isc_entry,&handle) != WSIO_OK) {
    return(ERROR);
}
wsio_cfg_in32(isc_entry,handle,offset,&data);

/* 'data' will now contain whatever was at 'offset' in */
/* configuration space */
```

**SEE ALSO**

`wsio_cfg_outXX(9F)`, `wsio_map_cfg_handle(9F)`, `wsio_unmap_cfg(9F)`
NAME

wsio_cfg_outXX - Macros for writing to configuration space.

SYNOPSIS

```c
#include <sys/wsio.h>

void wsio_cfg_outXX(
    struct isc_table_type *isc,
    wsio_addr_handle_t cfg_handle,
    uint32_t offset,
    uintXX_t *data
);
```

PARAMETERS

- **isc**: Pointer to the driver's `isc_table` entry.
- **cfg_handle**: Configuration handle.
- **offset**: Byte offset into the configuration space.
- **data**: Pointer to an appropriately sized and aligned memory space for the returned data.

DESCRIPTION

The `wsio_cfg_outXX` macros are called by device drivers to write to configuration space. The `cfg_handle` and the `offset` are used to specify the correct location to write to.

The value `XX` refers to 8, 16, 32, or 64, and indicates the amount of data to write to configuration space. Endian translation is performed automatically if host memory and the local bus have different endianness.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

```c
wsio_addr_handle_t handle;
uint32_t data = 0x5a;

if (wsio_map_cfg_handle(isc_entry,&handle) != WSIO_OK) {
    return(ERROR);
}
wsio_cfg_out32(isc_entry,handle,offset,data);

/* 0x5a will now be at 'offset' in configuration space */
```

SEE ALSO

`wsio_cfg_inXX(9F)`, `wsio_map_cfg_handle(9F)`, `wsio_unmap_cfg(9F)`
NAME

wsio_claim_node - Claim an I/O node.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_claim_node(
    void *handle,
    wsio_reg_node_t *node_info
);

PARAMETERS

handle Pointer to the handle of the I/O node to be claimed.
node_info Pointer to the wsio_reg_node_t structure that contains the claiming driver
information, including the type of node, the drv_info_t structure of the claiming
driver and callback function, argument, and event mask.

DESCRIPTION

A driver uses the wsio_reg_node WSIO service to claim an I/O node.

RETURN VALUES

WSIO_OK A new node was successfully claimed..
WSIO_INVALID_HNDL The caller specified an invalid handle.
WSIO_ERROR An error occurred.

CONSTRAINTS

None

EXAMPLE

None

SEE ALSO

wsio_async_scan(9F), wsio_destroy_legacy(9F), wsio_get_drv_priv(9F), wsio_get_legacy(9F),
wsio_node_get_isc(9F), wsio_put_drv_priv(9F), wsio_reg_legacy(9F), wsio_reg_node(9F),
wsio_reg_node_t(9S)
NAME

wsio_destroy_legacy - Destroy a legacy (SCSI-2) I/O node.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_destroy_legacy(
    void *handle,
    hw_path_t  *hw_path
);

PARAMETERS

handle Pointer to the handle of the node with which the legacy hardware path is registered by a previous call to wsio_reg_legacy.

hw_path Pointer to the hardware path of the legacy I/O node to be destroyed.

DESCRIPTION

The wsio_destroy_legacy WSIO service is used to destroy a legacy I/O node.

RETURN VALUES

WSIO_OK The legacy hardware path was successfully destroyed.
WSIO_INVALID_HNDL The caller specified an invalid handle parameter.
WSIO_INVALID_PARAM The caller specified invalid hw_path information.
WSIO_ERROR An error occurred.

CONSTRAINTS

None

SEE ALSO

wsio_async_scan(9F), wsio_claim_node(9F), wsio_get_drv_priv(9F), wsio_get_legacy(9F),
wsio_node_get_isc(9F), wsio_put_drv_priv(9F), wsio_reg_legacy(9F), wsio_reg_node(9F),
wsio_reg_node_t(9S)
wsio_dma_pass_thru(9F)

NAME

wsio_dma_pass_thru - Call a DMA pass-thru function that might not otherwise be accessible.

SYNOPSIS

#include <sys/wsio.h>

intptr_t wsio_dma_pass_thru(
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_pt_type_t pass_thru_type,
    intptr_t pass_thru_param
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
dma_handle DMA handle allocated using wsioAllocate_dma_handle.
pass_thru_type Indicates which pass-thru function to call. Two types are defined as shown in the following items:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_MAP_PTDEVICE_LOCK</td>
<td>Prevents a device's access to shared memory. This can be used so that processors can access host memory atomically, and can be used for synchronization. The parameter's pass_thru_param indicates whether shared memory is to be locked (1) or unlocked (0). Implementation of this function is not required, so the return code is zero (0) if the function is implemented or non-zero if not implemented.</td>
</tr>
<tr>
<td>WSIO_MAP_PTSYNC_BUS</td>
<td>Causes any FIFOs, buffers, or I/O caches associated with a device to be synchronized with memory. The parameter must be zero (0). This function returns zero (0) if the function is implemented and non-zero if it is not.</td>
</tr>
</tbody>
</table>

pass_thru_param Parameter that will get passed into the pass-thru function as an argument.

DESCRIPTION

The wsio_dma_pass_thru WSIO function is present to allow new DMA interfaces to be added without breaking binary compatibility. It allows such interfaces to be accessed if they are present.

RETURN VALUES

Dependent on the specific pass-thru function being called.

CONSTRAINTS

None
EXAMPLE

if (wsio_dma_pass_thru(isc_entry, dma_handle,
        WSIO_MAP_PT_SYNC_BUSSES, 0) != 0) {  
    /* The function is not implemented */
    return(ERROR);
} else {
    /* The function is implemented, and completed correctly */
    return(0);
}

/* This code will attempt to sync memory associated with
 * dma_handle
 */

SEE ALSO

wsio_allocate_dma_handle(9F), wsio_allocate_shared_mem(9F), wsio_fastmap_dma_buffer(9F),
wsio_flush_shared_mem(9F), wsio_free_dma_handle(9F), wsio_free_shared_mem(9F),
wsio_init_map_context(9F), wsio_iova_to_phys(9F), wsio_map_dma_buffer(9F),
wsio_remap_dma_buffer(9F), wsio_set_device_attributes(9F), wsio_set_dma_attributes(9F),
wsio_unmap_dma_buffer(9F)
NAME

wsio_dma_set_device_attributes - Associate DMA hints with a device.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_dma_set_device_attributes(
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_dma_attribute_t attribute,
    wsio_dma_attr_param_t param
);

PARAMETERS

isc Pointer to the driver's isc_table entry.

dma_handle DMA handle allocated using wsio_allocate_dma_handle.

attribute Indicates which hint to set for the device associated with dma_handle. The possible attributes are:

- **WSIO_DMA_ATTR_ADDR_WIDTH**: Bits of addressing supported by the device. This is used to determine whether a device can DMA directly to memory buffers.
  - Default value: 32

- **WSIO_DMA_ATTR_ALIGNMENT**: Byte alignment of DMA buffer required for device.
  - Default value: HW Dependent

- **WSIO_DMA_ATTR_ATM**: ATM hint. Used by hardware in some implementations.
  - 0 Not ATM
  - 1 ATM48 (optimize for 48-byte transfers)
  - 2 ATM192 (optimize for 192-byte transfers)
  - Default value: 0

- **WSIO_DMA_ATTR_CALLBACK**: Specifies a function to call when resources become available.
  - Default value: NULL

- **WSIO_DMA_ATTR_CALLBACK_ARG**: Specifies an argument to the callback function.
  - Default value: 0

- **WSIO_DMA_ATTR_FLUSH_ON_USE**: Specifies the cacheline must be flushed from any intermediate buffers as soon as it is referenced. This inhibits any coalescing of data by bus bridges.
  - Default value: 0
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_DMA_ATTR_IGN_ALIGN</td>
<td>Specifies the mapping service must not handle cacheline fragments in a special way.</td>
<td>0</td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_INTERLEAVE</td>
<td>IOVA allocation model</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 DMA streams are normally interleaved (mass-storage).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 DMA streams are normally not interleaved (networking).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 DMA buffers are static and accessed randomly (low fat).</td>
<td></td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_PREFETCH</td>
<td>Specifies how aggressively hardware is to prefetch for outbound DMA.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0 No prefetch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Moderate prefetch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 Aggressive prefetch</td>
<td></td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_SAFE</td>
<td>Specifies the use of the most conservative coherency model for inbound DMA.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Inhibits semicoherent transactions such as WRITE_PURGE unless it is guaranteed that no data in processor caches will be lost.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 On</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 Off</td>
<td></td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_TXN_SIZE</td>
<td>Specifies the default transaction size used by the device. This is used by hardware to optimize conversion of transactions between buses.</td>
<td>Hardware dependent</td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_INBOUND</td>
<td>DMA buffers will be used exclusively for inbound DMA.</td>
<td>0</td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_OUTBOUND</td>
<td>DMA buffers will be used exclusively for outbound DMA.</td>
<td>0</td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_STABLE</td>
<td>Indicates the buffer will not be modified by another entity while mapped for DMA.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>This is normally true (1) for data buffers, and false (0) for control structures.</td>
<td></td>
</tr>
</tbody>
</table>
Information dependent on the hint or attribute being set. Check the attribute list for more information.

DESCRIPTION

The `wsio_dma_set_device_attributes` function is used to associate DMA transaction hints and attributes with a specific device. These hints are overridden by any hints set for a specific DMA handle via `wsio_set_dma_attributes`, or some hints passed in as parameters to `wsio_map_dma_buffer`.

The `wsio_dma_set_device_attributes` function can be called in a non-blocking context. If `WSIO_DMA_ATTR_INTERLEAVE` is set to 1, a subsequent `wsio_allocate_shared_mem` or `wsio_map_dma_buffer` can only successfully request a buffer with a maximum of one page (4K) in length and this buffer cannot cross a page boundary. This is the limitation placed by the underlying platform. If a larger buffer is desirable, use the default value of 0. This larger buffer can be used for control structures rather than packet DMAs.

Do not call `wsio_dma_set_device_attributes` to set the `WSIO_DMA_ATTR_INTERLEAVE` to the default value of 0. The call will fail.

RETURN VALUES

- `WSIO_MAP_OK` Success
- `WSIO_MAP_E_PARAMETER_ERROR` Returned if an invalid parameter has caused failure of the call.

CONSTRAINTS

None

EXAMPLE

```c
if (wsio_dma_set_device_attributes(isc_entry, dma_handle, 
      WSIO_DMA_ATTR_INTERLEAVE, 1) != WSIO_MAP_OK) {
    /* There was a parameter error */
    return(ERROR);
} else {
    /* DMA streams are now not normally interleaved for all DMA 
     * associated with this device */
    return(0);
}
```

SEE ALSO

- `wsio_allocate_dma_handle(9F)`, `wsio_allocate_shared_mem(9F)`, `wsio_dma_pass_thru(9F)`, `wsio_fastmap_dma_buffer(9F)`, `wsio_free_dma_handle(9F)`, `wsio_free_shared_mem(9F)`, `wsio_flush_shared_mem(9F)`, `wsio_init_map_context(9F)`, `wsio_iova_to_phys(9F)`, `wsio_map_dma_buffer(9F)`, `wsio_remap_dma_buffer(9F)`, `wsio_set_dma_attributes(9F)`, `wsio_unmap_dma_buffer(9F)`
wsio_drv_data(9S)

NAME

wsio_drv_data - Driver-specific fields for WSIO drivers.

SYNOPSIS

#include <sys/wsio.h>

DESCRIPTION

The wsio_drv_data_t WSIO structure type, defined in <sys/wsio.h>, contains driver-specific fields for WSIO drivers.

STRUCTURE MEMBERS

Field | Purpose
--- | ---

**drv_path**

Follow these guidelines:

For device drivers, *drv_path* is typically a string that contain the interface card’s type and the device’s class. For example, *scsi_disk*.

For interface drivers, *drv_path* must match the card’s type. For example, *scsi*.

For pseudo drivers, *drv_path* must match the card’s class. For example, *graphics*.

**drv_type**

One of the following values:

T_INTERFACE The driver controls an interface card.

T_DEVICE The driver controls a hardware device.

**drv_flags**

One of the following values:

DRV_CONVERGED The driver meets the HP-UX Release 10.0 Converged I/O specifications. All new drivers must meet these specifications.

NOT_CONVERGED The driver conforms to the pre-Release 10.0 unconverged specifications.

**drv_minor_build**

Pointer to your minor number formatter. Use NULL if you do not provide one.

**drv_minor_decode**

Pointer to your minor number interpreter. Use NULL if you do not provide one.

EXAMPLES

```c
static wsio_drv_data_t sdisk_data = {
    "scsi_disk",
    T_DEVICE,
    DRV_CONVERGED,
    NULL,
    NULL,
};
```
NAME

wsio_drv_info - Structure containing pointers to other CDIO and WSIO data structures.

SYNOPSIS

#include <sys/wsio.h>

DESCRIPTION

The wsio_drv_info_t WSIO structure type, defined in <sys/wsio.h>, contains pointers to three other data structures.

STRUCTURE MEMBERS

- **drv_info**  Pointer to a drv_info_t CDIO structure.
- **drv_ops**  Pointer to a drv_ops_t CDIO structure.
- **drv_data**  Pointer to a wsio_drv_data_t structure.
- **driver_version**  Set to WSIO_DRV_CURRENT_VERSION.

SEE ALSO

drv_info(9S), drv_ops(9S), wsio_drv_data_t(9S)
wsio_fastmap_dma_buffer(9F)

NAME

wsio_fastmap_dma_buffer - Function to map an existing memory object for packet DMA.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_fastmap_dma_buffer(
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_range_type_t range_type,
    wsio_dma_map_t host_range,
    wsio_dma_map_t io_range
);

PARAMETERS

isc Pointer to the driver's isc_table entry.

dma_handle DMA handle allocated using wsio_allocate_dma_handle.

range_type Indicates the type of host memory being mapped. It can be:
    KERNELSPACE Indicates host_range is a kernel virtual buffer.
    PHYSICAL Indicates host_range is a physical buffer.
    > 0 Indicates host_range is in user space, and this will be the space ID of the virtual address.

host_range Pointer to an address/length structure that contains information about the host space to map. If the mapping was only partially completed, this will contain information about the remaining space to be mapped when the call completes.

io_range Pointer to an address/length structure that will contain information about the I/O space that was mapped.

DESCRIPTION

The wsio_fastmap_dma_buffer macro is called by a device driver to map an existing memory object for packet DMA. It operates in the same way as wsio_map_dma_buffer except the entire host address range must reside on a single physical page. If the range cannot be mapped in a single call, an error will be returned. Cacheline fragments are ignored (the same behavior as WSIO_DMA_IGN_ALIGNMENT).

The size of a buffer can have a maximum length of one page (4K) and this buffer can not cross a page boundary.

All mappings will remain in effect until wsio_unmap_dma_buffer or wsio_remap_dma_buffer are called to remove or change them.

Device drivers can set up a callback routine that will come into play if resources are not available at the time a mapping is attempted. If this callback is set up, WSIO_MAP_W_CALLBACK will be returned instead of a no resource error. When resources become available, the callback routine will be called to indicate this to the device driver. For more information on how to setup and use a callback, see the wsio_set_dma_callback manpage. wsio_fastmap_dma_buffer can be called in a non-blocking context.

RETURN VALUES

WSIO_MAP_OK Returned if the entire buffer has been mapped.
WSIO_MAP_W_CALLBACK  Returned if no resources are available and a callback function exists.
WSIO_MAP_E_NO_RESOURCES  Returned if no resources are available and no callback function exists.
WSIO_MAP_E_RESOURCE_ERROR  Returned if the request cannot and will never succeed.
WSIO_MAP_E_HIGH_ADDR  Returned if the call failed because the device cannot reach the destination address.
WSIO_MAP_E_PARAMETER_ERROR  Returned if an invalid parameter has caused failure of the call.
WSIO_MAP_E_UNKNOWN_ERROR  Returned for hardware or other errors.

CONSTRANTS
None

EXAMPLE

```c
void *dma_handle;
wsio_dma_map_t host_range, io_range;

dma_handle = wsio_allocate_dma_handle(isc_entry);

host_range.iov_base = host_virtual_address;
host_range.iov_len = dma_buffer_length;

if (wsio_fastmap_dma_buffer(isc_entry,dma_handle,
               KERNELSPACE,&host_range, &io_range)!= WSIO_MAP_OK) {
    /* Unable to map the range, so return an error */
    return(ERROR);
}

/* The host virtual buffer represented in the above code by
 * 'host_virtual_address' is now mapped.
 */
```

SEE ALSO

wsio_allocate_dma_handle(9F), wsio_allocate_shared_mem(9F), wsio_dma_pass_thru(9F),
wso_free_dma_handle(9F), wsio_free_shared_mem(9F), wsio_flush_shared_mem(9F),
wso_init_map_context(9F), wsio_iova_to_phys(9F), wsio_map_dma_buffer(9F),
wso_remap_dma_buffer(9F), wsio_set_device_attributes(9F), wsio_set_dma_attributes(9F),
wso_unmap_dma_buffer(9F)
wsio_flush_shared_mem(9F)

NAME

wsio_flush_shared_mem - Flush an I/O virtually contiguous DMA buffer.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_flush_shared_mem(
    struct isc_table_type *isc,
    void *dma_handle,
    size_t size,
    wsio_iova_t iova,
    wsio_vaddr_t vaddr,
    wsio_shared_mem_attr_t shared_mem_attr
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
dma_handle DMA handle allocated using wsio_allocate_dma_handle.
size Size of buffer to be flushed.
iova I/O virtual address of the shared memory.
vaddr Host virtual address of the shared memory.
shared_mem_attr Bitmask that was used to allocate the shared memory.

DESCRIPTION

The wsio_flush_shared_mem WSIO function is called by a device driver to guarantee the consistency of the memory object allocated via wsio_allocate_shared_mem. Any non-coherent buffers associated with the memory object are flushed. All parameters to the function must be the same as those passed to the call that allocated the memory.

The wsio_flush_shared_mem can be called in a non-blocking context.

RETURN VALUES

WSIO_MAP_OK Success
WSIO_W_NOP The call has no effect. The caller need not call it again.

CONSTRAINTS

None

EXAMPLE

void *dma_handle = NULL;
wsio_iova_t io_virtual_addr;
wsio_vaddr_t host_virtual_addr;

dma_handle = wsio_allocate_dma_handle(isc_entry);
if (dma_handle == NULL) {
    /* No handle allocated. */
    return(ERROR);
}

/* The DMA handle is now in the dma_handle variable */
if (wsio_allocate_shared_mem(
    isc_entry, dma_handle, buf_size, &io_virtual_addr,
&host_virtual_addr, 0) != WSIO_MAP_OK) {
  /* Unable to allocate the shared memory,
   * so return an error
   */
  return(ERROR);
}
/* A buffer of size 'buf_size' is now allocated and mapped in
 * both host virtual memory and I/O virtual memory space.
 */
wsio_flush_shared_mem(isc_entry, dma_handle, buf_size,
                      io_virtual_addr, host_virtual_addr, 0);

SEE ALSO

wsioAllocate_dma_handle(9F), wsioAllocate_shared_mem(9F), wsio_dma_pass_thru(9F),
wsio_free_dma_handle(9F), wsio_free_shared_mem(9F), wsio_iova_to_phys(9F),
wsio_set_device_attributes(9F), wsio_set_dma_attributes(9F)
NAME

wsio_free_dma_handle - Release a DMA handle.

SYNOPSIS

#include <sys/wsio.h>

void wsio_free_dma_handle(
    struct isc_table_type *isc,
    void *dma_handle
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
dma_handle Pointer to the DMA handle to free.

DESCRIPTION

The wsio_free_dma_handle WSIO function is called by device drivers to release a handle that has been allocated by wsio_allocate_dma_handle. It must be called anytime a handle is no longer needed.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

void *dma_handle;

dma_handle = wsio_allocate_dma_handle(isc_entry);
if (dma_handle == NULL) {
    /* No handle allocated. */
    return(ERROR);
}
/* The DMA handle is now in the dma_handle variable */

wsio_free_dma_handle(isc_entry, dma_handle);
/* The DMA handle has now been released */

SEE ALSO

wsio_allocate_dma_handle(9F), wsio_allocate_shared_mem(9F), wsio_dma_pass_thru(9F),
wsio_fastmap_dma_buffer(9F), wsio_free_shared_mem(9F), wsio_flush_shared_mem(9F),
wsio_init_map_context(9F), wsio_iova_to_phys(9F), wsio_map_dma_buffer(9F),
wsio_remap_dma_buffer(9F), wsio_set_device_attributes(9F), wsio_set_dma_attributes(9F)
wsio_unmap_dma_buffer(9F)
NAME

wsio_free_mem - Free memory allocated by wsio_alloc_mem.

SYNOPSIS

#include <sys/wsio.h>

void wsio_free_mem(
    wsio_mem_handle_t mem_handle,
    wsio_vaddr_t vaddr
);

PARAMETERS

mem_handle Handle allocated by a call to wsio_alloc_mem_handle.

vaddr Pointer to the allocated buffer.

DESCRIPTION

This WSIO service frees memory allocated by the service wsio_alloc_mem.

RETURN VALUES

None

CONSTRAINTS

May be called in user or interrupt context.

Do not call while holding a spinlock.

EXAMPLES

/* mem_handle is a handle allocated using
   wsio_alloc_mem_handle */
/* vaddr is a pointer to a block of memory allocated using
   wsio_alloc_mem */

wsio_free_mem(mem_handle, vaddr);

SEE ALSO

wsio_alloc_mem(9F), wsio_alloc_mem_handle(9F), wsio_free_mem_handle(9F)
wsio_free_mem_handle(9F)

NAME
wsio_free_mem_handle - Destroy handle allocated by wsio_alloc_mem_handle.

SYNOPSIS
#include <sys/wsio.h>
void wsio_free_mem_handle(
    wsio_mem_handle_t mem_handle
);

PARAMETERS
mem_handle Handle allocated by a call to wsio_alloc_mem_handle.

DESCRIPTION
This WSIO service destroys a mem_handle that was allocated by a previous call to
wsio_alloc_mem_handle.

RETURN VALUES
None

CONSTRAINTS
May be called in user or interrupt context.

EXAMPLES
/* mem_handle is a handle allocated using
   wsio_alloc_mem_handle() */

wsio_free_mem_handle(mem_handle);

SEE ALSO
wsio_alloc_mem(9F), wsio_alloc_mem_handle(9F), wsio_free_mem(9F).
wsio_free_shared_mem(9F)

NAME

wsio_free_shared_mem - Release an I/O virtually contiguous DMA buffer.

SYNOPSIS

#include <sys/wsio.h>

void wsio_free_shared_mem(
    struct isc_table_type *isc,
    void *dma_handle,
    size_t size,
    wsio_iova_t iova,
    wsio_vaddr_t vaddr,
    wsio_shared_mem_attr_t shared_mem_attr
);

PARAMETERS

isc          Pointer to the driver's isc_table entry.
dma_handle   DMA handle allocated using wsio_allocate_dma_handle.
size         Size of buffer to be released.
iova         I/O virtual address of the shared memory.
vaddr        Host virtual address of the shared memory.
shared_mem_attr  Bit mask that was used to allocate the shared memory.

DESCRIPTION

The wsio_free_shared_mem WSIO function is called by a device driver to release an I/O virtually contiguous DMA buffer that was allocated by wsio_allocate_shared_mem. All parameters to the macro must be the same as those passed to the call that allocated the memory. The wsio_free_shared_mem function can be called in a non-blocking context.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

void *dma_handle = NULL;
wsio_iova_t io_virtual_addr;
wsio_vaddr_t host_virtual_addr;

dma_handle = wsio_allocate_dma_handle(isc_entry);
if (dma_handle == NULL) {
    /* No handle allocated. */
    return(ERROR);
}

/* The DMA handle is now in the dma_handle variable */

if (wsio_allocate_shared_mem(isc_entry, dma_handle, buf_size,
    &io_virtual_addr,
    &host_virtual_addr, 0)
    != WSIO_MAP_OK) {
    /* Unable to allocate shared memory,
* so return an error */
    return(ERROR);
}

/* A buffer of size 'buf_size' is now allocated and mapped
 * in both host virtual memory and I/O virtual memory space.
 */
wsio_free_shared_mem(isc_entry, dma_handle, buf_size,
    io_virtual_addr, host_virtual_addr, 0);

SEE ALSO

wsio_allocate_dma_handle(9F), wsio_allocate_shared_mem(9F), wsio_dma_pass_thru(9F),
wsio_free_dma_handle(9F), wsio_flush_shared_mem(9F), wsio_iova_to_phys(9F),
wsio_set_device_attributes(9F), wsio_set_dma_attributes(9F)
wsio_get_active_processor_count(9F)

NAME

wsio_get_active_processor_count - Get the number of active CPUs in the system.

SYNOPSIS

#include <sys/wsio.h>

int wsio_get_active_processor_count(void);

PARAMETERS

None

DESCRIPTION

The wsio_get_active_processor_count WSIO function returns the number of CPUs currently active. On a running system the active CPU number may change. For example, if a processor is deallocated or allocated.

RETURN VALUES

The number of active processors.

CONSTRAINTS

None

EXAMPLE

int active_cpus;

active_cpus = wsio_get_active_processor_count();
printf("number of active CPUs %d\n", active_cpus);

SEE ALSO

wsio_get_processor_count(9F)
NAME

wsio_get_all_registers - Get an array of all available device registers.

SYNOPSIS

#include <sys/wsio.h>

wsio_reg_info_t * wsio_get_all_registers(
    struct isc_table_type *isc
);

PARAMETERS

isc    Pointer to the driver's isc_table entry.

DESCRIPTION

The wsio_get_all_registers WSIO function obtains an array of all of the registers for the device associated with the isc_table_type entry that is passed in. This function assumes that the device driver will know how many registers will be returned in the array and what their uses are.

RETURN VALUES

This function returns the address of an array of structures of type wsio_reg_info_t. If no registers exist, NULL will be returned.

CONSTRAINTS

None

EXAMPLE

wsio_reg_info_t *registers;

registers = wsio_get_all_registers(isc_entry);
if (registers == NULL) {
    /* No registers exist. Return an error. */
    return(ERROR);
}

/* All of the devices registers are now contained in the * 'registers' variable. They may be mapped as follows: */
if (wsio_map_reg(registers[1]) != WSIO_OK) {
    return(ERROR);
}

/* The second device register (index 1 into the array) will * now be mapped. */

SEE ALSO

wsio_map_reg(9F), wsio_read_regXX(9F), wsio_unmap_reg(9F), wsio_write_regXX(9F)
NAME

wsio_get_drv_priv - Retrieve private driver information from an I/O node.

SYNOPSIS

#include <sys/wsio.h>
wsio_ret_code_t wsio_get_drv_priv(
    void *handle,
    void **priv_info
);

PARAMETERS

handle Pointer to the handle of the I/O node from which to retrieve the private data.
priv_info Pointer to the private data.

DESCRIPTION

The wsio_get_drv_priv WSIO service is called to retrieve the driver private information
that was saved in the I/O node by a previous call to wsio_put_drv_priv. On return, priv_info
contains the handle of the data; the service does not copy it. The wsio_get_drv_priv routine
can be called in a non-blocking context.

RETURN VALUES

WSIO_OK The private information was successfully retrieved.
WSIO_INVALID_HNDL The caller specified an invalid I/O handle parameter.

CONSTRAINTS

None

EXAMPLE

wsio_legacy_info_t leg_node;

bzero((caddr_t)&leg_node, sizeof(wsio_legacy_info_t));
wsio_get_legacy(new_node, &leg_node);

while (leg_node->node != NULL)
{
    printf("name [%s]: desc[%s]\n", leg_node->name, leg_node->desc);
    wsio_get_legacy(new_node, &leg_node);
}

SEE ALSO

wsio_async_scan(9F), wsio_claim_node(9F), wsio_destroy_legacy(9F), wsio_node_get_isc(9F),
wsio_put_drv_priv(9F), wsio_reg_legacy(9F), wsio_reg_node(9F), wsio_reg_node_t(9S)
NAME

wsio_get_ioports - Obtain the addresses and sizes of I/O ports.

SYNOPSIS

#include <sys/wsio.h>

int wsio_get_ioports(
    struct isc_table_type *isc,
    int cnt,
    wsio_iop_t port_array[]
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
cnt Indicates the maximum number of ports to return.
port_array Array to store the ports in.

DESCRIPTION

The wsio_get_ioports WSIO function gets up to cnt I/O ports and stores them in an array. All ports will be obtained if the cnt variable is large enough. The driver must allocate enough space for all ports inside the port_array. Once these ports have been obtained, they can be mapped using wsio_map_port and accessed using wsio_port_inXX and wsio_port_outXX.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_ERROR There was a parameter error.

CONSTRAINTS

None

EXAMPLE

wsio_addr_handle_t port_handle;
wsio_iop_t ioports_array[10];
/* An array with enough space for all ports needs to */
/* be allocated */

if (wsio_get_ioports(isc_entry,10,ioports_array
    ) != WSIO_OK) {
    /* There was a problem obtaining the ports */
    return(ERROR);
}

if (wsio_map_port(isc_entry,ioports_array[0].addr,
    ioports_array[0].size, &port_handle
    ) != WSIO_OK) {
    /* There was an error mapping the port */
    return(ERROR);
}

/* Now unmmap the port */
if (wsio_unmap_port(isc_entry,ioports_array[0].addr,
    ioports_array[0].size, port_handle
    ) != WSIO_OK) {
    /* There was an error unmapping the port */
return(ERROR);
}

SEE ALSO

wsio_map_port(9F), wsio_port_inXX(9F), wsio_port_outXX(9F)
NAME

wsio_get_isc - Get the ISC structure pointer for a device file.

SYNOPSIS

#include <sys/io.h>
#include <sys/wsio.h>

int wsio_get_isc(
    dev_t dev,
    struct isc_table_type **isc_ptr,
    wsio_drv_info_t *wsio_drv_info
);

PARAMETERS

 dev The device file of the hardware for which we want an ISC pointer.
 isc_ptr A pointer to the location for the routine to put a pointer to the ISC structure.
 wsio_drv_info A pointer to the wsio_drv_info_t header structure of the driver, used to decode dev. If a NULL value is passed in this field, wsio_get_isc will use the wsio_drv_info_t structure of the character device (not block) whose major number matches that of the dev argument.

DESCRIPTION

The wsio_get_isc WSIO function finds an ISC table entry associated with the nearest interface ancestor of the device specified by dev, assuming that dev uses the HP-UX Converged I/O minor number format.

RETURN VALUES

1 Successful completion. The ISC pointer found is returned in isc_ptr.
0 Failure. The ISC could not be found.

CONSTRAINTS

None
NAME

wsio_get_legacy - Retrieve information about a legacy (SCSI-2) I/O node.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_get_legacy(
    void *handle,
    wsio_legacy_info_t *node_info
);

PARAMETERS

handle Pointer to the handle of the new (SCSI-3) node that has legacy I/O nodes.
node_info Pointer to the wsio_legacy_info_t structure of a previous legacy node.

DESCRIPTION

The wsio_get_legacy WSIO service is used to retrieve information on legacy I/O nodes associated with handle. This is a legacy I/O node registered with handle by a previous call to wsio_reg_legacy. There can be more than one legacy I/O node associated with handle. Therefore, this service can be called iteratively to return each legacy I/O node one at a time.

The caller passes in a pointer to a wsio_legacy_info_t structure with the information on the legacy node. The caller must initialize the following fields in the wsio_legacy_info_t structure:

version Set to WSIO_VERSION_0.
node Set to NULL to retrieve information for the first node. Otherwise, this is the handle of the last node found.

The caller must initialize the node field with NULL to retrieve the first legacy node. The service then finds the legacy I/O node and returns the wsio_legacy_info_t structure that contains the following information about the legacy node:

node The handle of the next legacy I/O node found.
hw_path The legacy hardware path.
type The legacy node type. Valid values are T_INTERFACE, T_VIRTBUS, and T_DEVICE.
drv_info The wsio_drv_info_t structure of the legacy driver.
instance The class instance number assigned to the legacy node.
name A pointer to a character string with the property name of the node.
desc A pointer to a character string with the property description of the node.
id The id bytes of the legacy IO node.

If there are more legacy I/O nodes the caller can continue to pass the structure in until the service returns a NULL value in the node field.

RETURN VALUES

WSIO_OK The information on the legacy hardware path was successfully returned.
WSIO_INVALID_HNDL The caller specified an invalid I/O handle parameter.
WSIO_INVALID_PARAM The caller specified invalid hw_path information.
WSIO_ERROR An error occurred.
CONSTRATN

None

EXAMPLE

wsio_legacy_info_t leg_node;

bzero((caddr_t)&leg_node, sizeof(wsio_legacy_info_t));
wsio_get_legacy(new_node, &leg_node);

while (leg_node->node != NULL)
{
    printf("name [%s]: desc[%s]\n", leg_node->name, leg_node->desc);
    wsio_get_legacy(new_node, &leg_node);
}

SEE ALSO

wsio_async_scan(9F), wsio_claim_node(9F), wsio_destroy_legacy(9F), wsio_get_drv_priv(9F),
wsio_node_get_isc(9F), wsio_put_drv_priv(9F), wsio_reg_legacy(9F), wsio_reg_node(9F),
wsio_reg_node_t(9S)
NAME

wsio_get_processor_count - Get the number of CPUs in the system.

SYNOPSIS

#include <sys/wsio.h>

int wsio_get_processor_count(void);

PARAMETERS

None

DESCRIPTION

The wsio_get_processor_count WSIO function returns the number of CPUs configured into the system. On a running system the number of CPUs may change. For example, if a processor is deallocated or allocated.

RETURN VALUES

The number of processors.

CONSTRAINTS

None

EXAMPLE

int num_cpus;

num_cpus = wsio_get_processor_count();

printf("number of CPUs %d\n", num_cpus);

SEE ALSO

wsio_get_active_processor_count(9F)
wsio_get_relationship(9F)

NAME

wsio_get_relationship - Returns the specified relative in the hierarchical tree.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_get_relationship (  
    IN struct isc_table_type *isc,  
    IN wsio_relationship_t which_relation,  
    IN struct isc_table_type *prev,  
    OUT struct isc_table_type **next
);

PARAMETERS

isc The isc handle of the HBA to be queried.
which_relation The relation to get, PARENT or CHILD.
prev The isc handle of the last relation to be returned.
next When called, a pointer to an isc handle. On return, the isc handle of the next member in the relationship.

DESCRIPTION

This service is called to get the parents or children of an interface. The first parameter is the isc handle of the interface to query. The which_relation parameter indicates what relationship to query. Valid values are WSIO_ISC_GET_PARENT or WSIO_ISC_GET_CHILD. The prev parameter can be used to indicate the last relation returned if the isc has multiple parents or children. If prev is NULL, the isc of the first child or parent is returned; else, the isc of the next child or parent is returned. The caller must make no assumptions about the order in which members are returned. The last parameter is where the isc of the next relation is returned.

In the following example the caller gets the isc of each if the children of an HBA.

RETURN VALUES

WSIO_OK The next isc is returned.
WSIO_ERROR An error occurred

CONSTRAINTS

None

EXAMPLE

struct isc_table_type *prev, *next;
struct isc_table_type hba_isc;

wsio_ret_code_t ret;
prev = next = NULL;
hba_isc = my_hba_isc;
do {
    /* Get the interface child */
    ret = wsio_get_relationship ( hba_isc, WSIO_ISC_GET_CHILD, prev, &next);
    ...
    /* Set the prev to next, if intended to get another child */
    prev = next;
} while (ret == WSIO_OK);

SEE ALSO

440 WSIO Reference Pages
NAME

wsio_get_system_params - Get information about the system.

SYNOPSIS

#include <sys/wsio.h>

int wsio_get_system_params_t *wsio_get_system_params(
    struct isc_table_type *isc,
    wsio_get_sys_parm_t id,
    wsio_unintptr_t *parm
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
id An identifier indicating what parameter to get.
parm The value returned will be written to a memory location pointed to by parm.
id parm

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_CACHELINE_SIZE</td>
<td>Indicates the cacheline size in bytes</td>
</tr>
<tr>
<td>WSIO_DEFAULT_PAGE_SIZE</td>
<td>Indicates the default page size in bytes.</td>
</tr>
<tr>
<td>WSIO_DMA_64BIT_ADDRESSING</td>
<td>Indicates 64-bit addressing capability (1), or not capable (0).</td>
</tr>
<tr>
<td>WSIO_DMA_COHERENT_IO</td>
<td>Indicates I/O coherent (1), or not coherent (0).</td>
</tr>
<tr>
<td>WSIO_DMA_IOPDIR_PRESENT</td>
<td>Indicates IOPDIR is present (1) or not present (0).</td>
</tr>
<tr>
<td>WSIO_NUM_CPUS</td>
<td>Indicates the number of CPUs on the system.</td>
</tr>
</tbody>
</table>

DESCRIPTION

The wsio_get_system_params WSIO function obtains all system parameters that are currently defined. The id is to identify what parameter to retrieve. The value that the caller is interested in will be stored into the memory location pointed to by parm.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_ERROR Error

CONSTRAINTS

None

EXAMPLE

int ret;
wsio_unintptr_t value;

ret = wsio_get_system_params(isc_entry,
    WSIO_CACHELINE_SIZE, &value);

/* The cacheline size will be obtained and put into value */
**NAME**

wsio_hwpath_to_isc - Returns the HBA at the specified hardware path.

**SYNOPSIS**

```c
#include <sys/wsio.h>

wsio_ret__code_t wsio_hwpath_to_isc(
    IN struct isc_table_type *ancestor,
    IN char *path,
    IN/OUT struct isc_table_type **isc
);
```

**PARAMETERS**

- `ancestor`: Either NULL or the isc handle of an ancestor to start the hw_path from.
- `path`: A string with the hardware path.
- `isc`:
  - IN: A pointer to an uninitialized isc handle.
  - OUT: The handle initialized with the isc of the target.

**DESCRIPTION**

This service is used to find the isc structure of an interface at the path specified by the `isc` and `path` parameters. If the `ancestor` parameter is not NULL, `path` is assumed to be relative to the hardware path associated with the `isc`; otherwise, `path` is treated as an absolute path. If successful, the `isc` handle of the interface is returned. An interface can be of type WSIO_INTERFACE, WSIO_VIRT_BUS, and WSIO_TRANS.

**RETURN VALUES**

- **WSIO_OK**: The isc handle is returned.
- **WSIO_ERROR**: An error occurred.

**CONSTRAINTS**

None

**SEE ALSO**

wsio_isc_to_hwpath(9F)
wsio_init_map_context(9F)

NAME

wsio_init_map_context - Initialize the context used for DMA mapping.

SYNOPSIS

#include <sys/wsio.h>

void wsio_init_map_context(
    wsio_map_context_t *context
);

PARAMETERS

context Pointer to the context to be initialized.

DESCRIPTION

The wsio_init_map_context WSIO macro is called by device drivers to initialize a context that is needed by wsio_map_dma_buffer and other map related functions. A context is used internally by mapping services so that system resources can be used efficiently. The same context must be used for a set of DMA mappings that are all going to be unmapped at the same time. This context must be initialized before its first use. If the mapping that is being performed is independent of all other mappings, a context is not necessary and NULL must be passed to the mapping function in the context field.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

wsio_map_context_t dma_context;

wsio_init_map_context(&dma_context);
/* dma_context is now ready to be used for mapping. It can
 * be used for any number of mappings as long as they are
 * all unmapped at the same time
 */

SEE ALSO

wsio_allocate_dma_handle(9F), wsio_dma_pass_thru(9F), wsio_fastmap_dma_buffer(9F),
wsio_free_dma_handle(9F), wsio_iova_to_phys(9F), wsio_map_dma_buffer(9F),
wsio_remap_dma_buffer(9F), wsio_set_device_attributes(9F), wsio_set_dma_attributes(9F),
wsio_unmap_dma_buffer(9F)
NAME

wsio_install_driver - Install a driver's header structure into the WSIO CDIO.

SYNOPSIS

#include <sys/wsio.h>

int wsio_install_driver(
    void *wsio_drv_info
);

PARAMETERS

wsio_drv_info Pointer to the driver's wsio_info_t structure.

DESCRIPTION

The wsio_install_driver WSIO function installs a driver's header structure into the WSIO CDIO.

RETURN VALUES

1 Successful completion.
0 Error. The major number specified for the driver is already in use. The following message is displayed on the system console and in the error log file:
    wsio_install_driver: Install of driver driver failed.

CONSTRAINTS

None

EXAMPLES

/* Declare the driver entry points */
static drv_ops_t    beep_ops = {
    beep_open,  /* open */
    beep_close, /* close */
    NULL,       /* strategy */
    NULL,       /* dump */
    NULL,       /* psize */
    NULL,       /* mount */
    NULL,       /* read */
    NULL,       /* write */
    beep_ioctl, /* ioctl */
    NULL,       /* select */
    NULL,       /* option_1 */
    NULL,       /* reserved1 */
    0,          /* d_psize1 */
    0,          /* d_drv_cb */
    NULL,       /* reserved4 */
    0           /* flag */
};

/* Declare the CDIO driver-specific fields */
/* Flags DRV_CHAR/DRV_BLOCK/DRV_PSEUDO/DRV_SCAN/DRV_MP_SAFE/ 
   DRV_SAFE_CONF */
static drv_info_t   beep_info = {
    "beep",         /* char *name for device type */
    "graf_pseudo",  /* char *name for device class */
    DRV_PSEUDO|DRV_CHAR,/*ubit32 flags pseudo? block? char? scan? */
    -1,             /* int b_major  maj dev# if block type */
    168,            /* int c_major maj dev# if char type */
NULL, /* struct cdio *cdio drivers set to NULL */
NULL, /* void *gio_private drivers set to NULL */
NULL /* void *cdio_private drivers set to NULL */

/* Declare the WSIO driver-specific fields */
static wsio_drv_data_t beep_data = {
    "hil",    /* char *drv_path   match probes-drivers */
    T_DEVICE, /* sbit8 drv_type   type of H/W: dev or IF */
    DRV_CONVERGED,
    /* ubit32 drv_flags DRV_CONV... or NOT_C... */
    NULL,
    /* int (*drv_minor_build)() minor# formatter */
    NULL
    /* int (*drv_minor_decode)() interpreter */
};
static wsio_drv_info_t beep_wsio_info = {
    &beep_info, /* drv_info_t *drv_info */
    &beep_ops, /* drv_ops_t *drv_ops driver entry points */
    &beep_data /* wsio_drv_data_t *drv_data */
};

void beep_install()
{
    /* register driver with WSIO */
    wsio_install_driver( beep_wsio_info );
}

SEE ALSO

install_driver(9F)
wsio_install_drv_event_handler(9F)

NAME

wsio_install_drv_event_handler - Install a driver's event handler.

SYNOPSIS

#include <sys/wsio.h>

int wsio_install_drv_event_handler(
    wsio_drv_info_t *drv_info,
    void (*drv_handler)(wsio_generic_event_t *generic_ptr)
);

PARAMETERS

drv_info Pointer to the driver's wsio_drv_info_t structure.

drv_handler Function pointer to a driver's event handler

DESCRIPTION

The wsio_install_drv_event_handler WSIO function is called by a device driver to register an event handler to deal with events. When an event occurs, such as a suspend request, WSIO will invoke this handler. This function must be called in a driver's installation routine after wsio_install_driver is executed.

RETURN VALUES

WSIO_OK Returned on success.
WSIO_DRV_NOT_FOUND An earlier call to wsio_install_driver was not successful.
WSIO_HANDLER_NULL drv_handler is a NULL pointer.
WSIO_INFO_NULL drv_info is a NULL pointer.

CONSTRAINTS

None

EXAMPLE

static wsio_drv_info_t my_drv_info {
    &my_info,
    &my_ops,
    &my_data,
    WSIO_DRV_CURRENT_VERSION,
    /* This is a macro defined in wsio.h, bearing a version
     * stamp */
}

void my_driver_install() {
    if (wsio_install_driver(&my_drv_info) != 1) {
        return;
    }
    /* Register the event handler with WSIO */
    if (wsio_install_drv_event_handler(
            &my_drv_info, my_handler) != WSIO_OK) {
        /* The example given shows that wsio_uninstall_driver()
         * is called. In this case, the driver
         * will not be brought up. If a driver chooses to
         * ignore the error and proceed, it will
         * run without being able to handle events */
        wsio_uninstall_driver(&my_drv_info);
        return;
    }
}
SEE ALSO

wsio_query_supported_function(9E), wsio_reg_drv_capability_mask(9F),
wsio_uninstall_drv_event_handler(9F)
NAME

wsio_intr_activate - Enable an interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_activate(
    struct isc_table_type *isc,
    wsio_intr_object_t obj
);

PARAMETERS

isc Pointer to the driver's isc_table entry.

obj Interrupt object to enable.

DESCRIPTION

The `wsio_intr_activate` WSIO function activates an interrupt object that was allocated with `wsio_intr_alloc`. The interrupt object must be activated before the system will call the device driver's ISR (as specified in `wsio_intr_alloc`). It is assumed that (if possible) the device will not generate interrupts until after this function is called. The `wsio_intr_deactivate` or `wsio_intr_deactivate_nowait` service undoes the effects of this function.

Attempting to activate an interrupt object that is already active is an error condition that returns WSIO_INTR_ACTIVATED, without modifying the interrupt object.

RETURN VALUES

WSIO_OK Operation succeeded.

WSIO_ERROR Failure. Could not enable `obj`.

WSIO_INTR_INV_OBJ Must call `wsio_intr_set_cpu_spec` or `wsio_intr_set_irq_line` first.

WSIO_INTR_ACTIVATED `obj` already active.

WSIO_PARM_ERROR Invalid parameters.

CONSTRAINTS

None

EXAMPLE

/* Allocate a line based interrupt and activate it */
wsio_intr_object_t obj;
int status;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
        (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
        WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

SEE ALSO

wsio_intr_alloc(9F), wsio_intr_deactivate(9F), wsio_intr_deactivate_nowait(9F),
wsio_intr_set_cpu_spec(9F), wsio_intr_set_line(9F)
NAME

wsio_intr_alloc - Allocate an interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_alloc(
    struct isc_table_type *isc,
    wsio_drv_isr_t isr,
    uintptr_t arg,
    uint64_t flags,
    wsio_intr_object_t *obj
);

PARAMETERS

isc   Pointer to the driver's isc_table entry.
isr   Address of the interrupt service routine.
arg   Argument to be passed to driver_isr.
flags Shared or exclusive flag.
obj   Interrupt object.

DESCRIPTION

The wsio_intr_alloc function allocates and initializes an interrupt object that will field interrupts generated by the given device associated with the isc. This routine may sleep and thus must be called from a thread context. This routine might allocate hardware resource; use it with care.

The flags parameter must be either WSIO_INTR_EXCLUSIVE if the device driver's ISR cannot be shared, or zero if the ISR can be shared. If the WSIO_INTR_EXCLUSIVE flag is not present, the driver's ISR may be called even if the device did not generate an interrupt. If the device driver has no way of determining if the card needs servicing, the device driver's ISR cannot be shared. For such a card, running the device driver's ISR indicates the card needs servicing.

RETURN VALUES

WSIO_OK      Successful completion.
WSIO_ERROR   Failure; no interrupt services available.
WSIO_INTR_INV_FLAG Must specify 0 (shared) or WSIO_INTR_EXCLUSIVE only.
WSIO_MEM_ALLOC_FAILED Interrupt services failed to allocate memory.
WSIO_PARM_ERROR Invalid parameters

CONSTRAINTS

None

EXAMPLE

/* Allocate a line based interrupt */
wsio_intr_object_t obj;
int status;

    /* allocate an interrupt object for a shared interrupt */
    status = wsio_intr_alloc(isc, isr,
                                 (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

SEE ALSO

wsio_intr_free(9F)
NAME

wsio_intr_assign_cpus - Request CPU assignments for interrupt locality vectors.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_assign_cpus(
    struct isc_table_type *isc,
    int cnt,
    intr_loc_t *loc_list,
    spu_t *cpu_list
);

PARAMETERS

isc Pointer to the isc structure of the interface card.

cnt The number of entries in loc_list and cpu_list.

loc_list Pointer to an array of cnt interrupt locality vector entries.

cpu_list Pointer to an array of cnt uninitialized spu_t data structures.

DESCRIPTION

The wsio_intr_assign_cpus service is called by drivers to assign CPUs to each interrupt
locality vector in the loc_list passed in. The cnt and loc_list parameters are returned by a call to
wsio_intr_get_loc. The cpu_array is an array of cnt spu_t data structures. The service assigns
CPUs to each entry in the array based on the locality assignments in the loc_list. For example,
cpu_list[x] is assigned a CPU according to the locality in loc_list[x].loc. On successful
completion, the cpu_list array is initialized with spu_t assignments.

It is expected that only drivers using MSI-X type interrupts that require a large number of interrupt
resources per card will use this service.

RETURN VALUES

WSIO_OK The operation was successful.

WSIO_ERROR The operation failed.

EXAMPLES

#define MAX_VECTORS 64
my_msi_driver_init() {
    intr_loc_t *my_loc;
    spu_t  cpu_list[MAX_VECTORS];
    wsio_msi_set_info_t msi_args[MAX_VECTORS];
    int vec_cnt = MAX_VECTORS;

    if (wsio_msi_capability(isc, WSIO_MSI_X_CAPABILITY, vec_cnt) != WSIO_OK)
        return ERROR;

    if ((wsio_intr_req_loc(my_isc, vec_cnt, WSIO_INTR_ALL_LOC)) != WSIO_OK)
        return ERROR;

    if ((wsio_intr_get_loc(my_isc, &vec_cnt, &my_loc)) != WSIO_OK)
        return (ERROR);

    if ((wsio_intr_assign_cpus( my_isc, vec_cnt, my_loc, cpu_list )) != WSIO_OK)
        return(ERROR);
if (wsio_msi_alloc(isc,WSIO_MSI_X_TYPE,vec_cnt,0,&msi_obj) != WSIO_OK)
    return ERROR);

for (i = 0; i < vec_cnt; i++)
{
    msi_args[i]->mask = WSIO_MSI_SET_CPU | WSIO_MSI_SET_ISR;
    msi_args[i]->isr   = my_drv_isr;
    msi_args[i]->isr_arg = isc;
    msi_args[i]->cpu_id  = my_intr_vec->cpu;
}

msi_hints = WSIO_MSI_ASSIGN_RANGE;
if(wsio_msi_assign(msi_obj,msi_hints,msi_args,0,vec_cnt)) == WSIO_OK)
{
    msi_hints = WSIO_MSI_VECTOR_RANGE;
    ret =  wsio_msi_enable(msi_obj, msi_hints, 0, vec_cnt);
}

SEE ALSO

wsio_intr_get_loc(9F), wsio_intr_req_loc(9F)
wsio_intr_deactivate(9F)

NAME

wsio_intr_deactivate - Disable an interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_deactivate

    (struct isc_table_type *isc,
     wsio_intr_object_t obj)
;

PARAMETERS

isc Pointer to the driver's isc_table entry.

obj Interrupt object.

DESCRIPTION

The wsio_intr_deactivate WSIO function deactivates an interrupt object that was previously activated with wsio_intr_activate. By deactivating the interrupt object the system will stop calling the device driver's ISR (as specified in wsio_intr_alloc). It is assumed that (if possible) the device's interrupts will be disabled before this function is called. This function will sleep.

Attempting to deactivate an interrupt object that has not been activated is an error condition that returns WSIO_INTR_DEACTIVATED, without modifying the interrupt object.

RETURN VALUES

WSIO_OK Successful completion.

WSIO_INTR_DEACTIVATED obj not active.

WSIO_PARM_ERROR Invalid parameters.

WSIO_ERROR Could not deactivate obj.

CONSTRAINTS

Do not call in an interrupt context.

EXAMPLE

/* Allocate a line based interrupt and activate it */

wsio_intr_object_t obj;
int status;

/* allocate an interrupt object */

status = wsio_intr_alloc(isc, driver_isr,(uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a Level Sensitive IRQ */

status = wsio_intr_set_irq_line(isc, obj,
    WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */

status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

.............

/* deactivate the interrupt */
status = wsio_intr_deactivate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

**SEE ALSO**

*wsio_intr_activate*(9F), *wsio_intr_alloc*(9F), *wsio_intr_deactivate_nowait*(9F), *wsio_intr_set_cpu_spec*(9F), *wsio_intr_set_line*(9F),
NAME

wsio_intr_deactivate_nowait - Disable an interrupt object with callback.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_deactivate_nowait(
    struct isc_table_type *isc,
    wsio_intr_object_t obj,
    wsio_intr_deact_cb_t callback_func,
    uintptr_t arg
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
obj Interrupt object.
callback_func A driver callback function that will be executed.
arg Parameter passed back to a driver in the callback.

DESCRIPTION

The wsio_intr_deactivate_nowait WSIO function is a mirror of wsio_intr_deactivate except that this one does not sleep. It deactivates an interrupt object that was previously activated with wsio_intr_activate. By deactivating the interrupt object the system will stop calling the device driver's ISR (as specified in wsio_intr_alloc). It is assumed that, if possible, the device's interrupts will be disabled before this function is called.

Attempting to deactivate an interrupt object that has not been activated is an error condition that returns WSIO_INTR_DEACTIVATED, without modifying the interrupt object.

This function will return immediately. Upon receiving confirmation that the interrupts have been disabled, the callback_func will be executed passing the arg back to the driver. Since this call will not sleep, it can be called in a non-blocking context.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_INTR_DEACTIVATED obj not active.
WSIO_ERROR Operation failed.
WSIO_INTR_INV_OBJ Must call wsio_intr_set_cpu_spec or wsio_intr_set_irq_line first.
WSIO_NO_INTR_CB No call back function passed in.
WSIO_PARM_ERROR Invalid parameters.

CONSTRAINTS

None

EXAMPLE

/* Driver's callback function for wsio_intr_deactivate_nowait() */
void my_callback(struct isc_table_type *isc)
{
    ...
}
/ * Allocate a line based interrupt and activate it */
wsio_intr_object_t obj;
int status;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
    (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
    WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* deactivate the interrupt */
status = wsio_intr_deactivate_nowait(isc,
    obj, my_callback, isc);
if (status != WSIO_OK) {
    return(ERROR);
}

SEE ALSO
wsio_intr_activate(9F), wsio_intr_alloc(9F), wsio_intr_deactivate(9F), wsio_intr_set_cpu_spec(9F),
wsio_intr_set_line(9F)
NAME

wsio_intr_free - Free an interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_free(
    struct isc_table_type *isc,
    wsio_intr_object_t obj
);

PARAMETERS

isc  Pointer to the driver's isc_table entry.
obj  Interrupt object.

DESCRIPTION

Before this function returns, the services guarantee that all outstanding ISRs have been run to
completion. The obj object must not be active (either never activated, or deactivated with
wsio_intr_deactivate or wsio_intr_deactivate_nowait) when this function is called.
This function may block, so it must be called in a thread context.

RETURN VALUE

WSIO_OK         Successful completion.
WSIO_INTR_ACTIVATED Must call wsio_intr_deactivate or
                    wsio_intr_deactivate_nowait first.
WSIO_PARM_ERROR Invalid parameters.

CONSTRAINTS

Do not call in an interrupt context.

EXAMPLE

/* Allocate a line based interrupt and activate it */
wsio_intr_object_t obj;
int status;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
            (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
    WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* deactivate the interrupt */
status = wsio_intr_deactivate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}
/* free the interrupt obj */
status = wsio_intr_free(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}
obj = NULL;

SEE ALSO

wsio_intr_alloc(9F)
NAME

wsio_intr_get_assigned_cpu - Get the currently assigned CPU for the interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_get_assigned_cpu(
    wsio_intr obj,
    intptr_t *cpu_spec
);

PARAMETERS

obj Interrupt object.

cpu_spec CPU specification.

DESCRIPTION

The wsio_intr_get_assigned_cpu WSIO function returns the currently assigned CPU for the passed interrupt object. The return type for cpu_spec will depend on platform. On PA, it will return assigned CPU in the location pointed by cpu_spec.

The INTR_ATTR_ASSIGNED_CPU attribute is not currently defined in the BN-CDIO specification, but will be added as it is a necessary functionality.

RETURN VALUES

WSIO_OK Successful completion.

WSIO_ERROR INTR_ATTR_ASSIGNED_CPU not implemented in CDIO.

WSIO_PARM_ERROR Invalid parameters.

CONSTRAINTS

None

EXAMPLE

/* Allocate a transaction based interrupt */
wsio_intr_object_t obj;
int status;
intptr_t cpu_spec;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
    (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a TXN based interrupt */
status = wsio_intr_set_cpu_spec(isc, obj, WSIO_INTR_CPU_ANY);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* get the CPU spec */
status = wsio_intr_get_assigned_cpu(obj, &cpu_spec);
if (status != WSIO_OK) {
    return(ERROR);
}
printf ("assigned cpu for the interrupt object is = %d\n", 
    cpu_spec);

SEE ALSO

wsio_intr_activate(9F), wsio_intr_alloc(9F), wsio_intr_set_cpu_spec(9F)
NAME

wsio_intr_get_attribute - Get the attribute value for an interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_get_attribute(
    IN struct isc_table_type *isc,
    IN wsio_intr_object_t *iobj,
    IN wsio_intr_attrib_t attrib,
    OUT intptr_t attr_val
);

PARAMETERS

isc        Pointer to the ISC table entry for the driver instance.
attrib     Type of interrupt attribute to query.
iobj       WSIO interrupt object.
attr_val    Current value of the attribute.

DESCRIPTION

This service returns the current value of an interrupt attribute for the specified interrupt object. Valid interrupt attributes that can be queried include:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value Returned/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_INTR_ATTR_ENABLED</td>
<td>WSIO_INTR_ATTR_VAL_SET, interrupt is enabled; WSIO_INTR_ATTR_VAL_CLEAR, interrupt is disabled.</td>
</tr>
<tr>
<td>WSIO_INTR_ATTR_ORDERED</td>
<td>WSIO_INTR_ATTR_VAL_SET, interrupts are ordered with respect to DMA; WSIO_INTR_ATTR_VAL_CLEAR, interrupts are not ordered with respect to DMA.</td>
</tr>
<tr>
<td>WSIO_INTR_ATTR_LEVEL</td>
<td>WSIO_INTR_EDGE_SENSITIVE, interrupt is edge triggered; WSIO_INTR_LEVEL_SENSITIVE, interrupt is level triggered.</td>
</tr>
<tr>
<td>WSIO_INTR_ATTR_SHARE_VEC</td>
<td>WSIO_INTR_SHARED, driver's ISR is shared; WSIO_INTR_EXCLUSIVE, driver's ISR is not shared.</td>
</tr>
</tbody>
</table>

This interface can be called from a non-blocking context.

RETURN VALUES

WSIO_OK          Attribute was successfully changed.
WSIO_INVALID_ISC Not a valid ISC structure.
WSIO_INTR_INV_OBJ Not a valid interrupt object.
WSIO_NOT_IMPLEMENTED Specified attribute can not be queried.
WSIO_PARM_ERROR  The attrib parameter is not valid.

CONSTRAINTS

None

EXAMPLES

wsio_intr_object_t iobj;
wsio_intr_attrib_t attrib;
intptr_t attr_val;

attrib = WSIO_INTR_ATTR_LEVEL;
/* Find out if the interrupt associated with iobj is edge
* or level triggered.
*/
status = wsio_intr_get_attribute (isc, iobj, attrib,
    &attr_val);

SEE ALSO

wsio_intr_alloc(9F), wsio_intr_free(9F), wsio_intr_activate(9F), wsio_intr_deactivate(9F),
wsio_intr_deactivate_nowait(9F), wsio_intr_set_cpu_spec(9F), wsio_intr_set_irq_line(9F),
wsio_intr_set_attribute(9F)
wsio_intr_get_irq_line(9F)

NAME

wsio_intr_get_irq_line - Get the interrupt line number.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_get_irq_line(
    struct isc_table_type *isc,
    wsio_intr_object_t obj,
    intptr_t *irq_line_num
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
obj Interrupt object.
irq_line_num The interrupt line number.

DESCRIPTION

The wsio_intr_get_irq_line() WSIO function returns the line number that the given obj is currently using. In most cases this routine is not necessary, as WSIO_IRQ_LINE_AUTO can be given to wsio_intr_set_irq_line. However, if for some reason a device driver needs to know the interrupt line that a card is to use, this function provides the necessary data. On a running system the line number assigned to an obj may change. For example, when a processor is deallocated, the driver services will reassign the interrupt line on behalf of the driver.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_INTR_INV_OBJ Must call wsio_intr_set_irq_line first.
WSIO_INTR_ACTIVATED Object not active, call wsio_intr_activate first.
WSIO_ERROR Failed to get line number.
WSIO_PARM_ERROR Invalid parameters.

CONSTRAINTS

None

EXAMPLE

/* Allocate a line based interrupt and activate it */
wsio_intr_object_t obj;
int status;
intptr_t irq;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
    (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
    WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
    return(ERROR);
}
/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* get the line number */
status = wsio_intr_get_irq_line(isc, obj, &irq);
if (status != WSIO_OK) {
    return(ERROR);
}
printf ("my line number is %ld\n", irq);

SEE ALSO

wsio_intr_activate(9F), wsio_intr_alloc(9F), wsio_intr_set_line(9F)
NAME

wsio_intr_get_loc - Retrieve interrupt vector assignments.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_get_loc(
    struct isc_table_type *isc,
    int *cnt,
    intr_loc_t *loc_list
);

PARAMETERS

isc Pointer to the isc structure of the interface card.
cnt IN: Pointer to an integer. OUT: The number of vector entries in loc_list.
loc_list Pointer to an array of interrupt locality vectors.

DESCRIPTION

The wsio_intr_get_loc service retrieves a pointer to the cached interrupt or locality vectors. Each vector consists of an interrupt ID (intr_id) and locality (loc). You must call wsio_intr_req_loc before calling this service. Otherwise, wsio_intr_get_loc returns WSIO_ERROR. This service does not allocate any information; it only returns the cached information. Therefore, it can be called multiple times to retrieve the vectors.

The caller passes in the addresses of both an intr_loc_t pointer, loc_list, and an integer (cnt). If successful, the service returns the number of locality vectors in the cnt parameter and the array of locality vectors in loc_list.

The intr_id indicates the vector number. For example, if three vectors are returned, the intr_ids are 0, 1, 2.

RETURN VALUES

WSIO_OK The operation was successful.
WSIO_ERROR The operation failed.

EXAMPLES

#define MAX_VECTORS 64
my_msi_driver_init()
{
    intr_loc_t *my_loc;
    spu_t cpu_list[MAX_VECTORS];
    wsio_msi_set_info_t msi_args[MAX_VECTORS];
    int vec_cnt = MAX_VECTORS;

    if (wsio_msi_capability(isc, WSIO_MSI_X_CAPABILITY, vec_cnt) != WSIO_OK)
        return ERROR;

    if (((wsio_intr_req_loc(my ISC, vec_cnt, WSIO_INTR_ ALL_LOC)) != WSIO_OK)
        return (ERROR);

    if (((wsio_intr_get_loc(my ISC, &vec_cnt, &my_loc)) != WSIO_OK)
        return (ERROR);

    if (((wsio_intr_assign_cpus( my ISC, vec_cnt, my_loc, cpu_list)) != WSIO_OK)
        return ERROR);

466  WSIO Reference Pages
if (wsio_msi_alloc(isc, WSIO_MSI_X_TYPE, vec_cnt, 0, &msi_obj) != WSIO_OK)
    return ERROR;

for (i = 0; i < vec_cnt; i++)
{
    msi_args[i]->mask = WSIO_MSI_SET_CPU | WSIO_MSI_SET_ISR;
    msi_args[i]->isr  = my_drv_isr;
    msi_args[i]->isr_arg = isc;
    msi_args[i]->cpu_id  = my_intr_vec->cpu;
}

msi_hints = WSIO_MSI_ASSIGN_RANGE;
if(wsio_msi_assign(msi_obj, msi_hints, msi_args, 0, vec_cnt)) == WSIO_OK)
{
    msi_hints = WSIO_MSI_VECTOR_RANGE;
    ret = wsio_msi_enable(msi_obj, msi_hints, 0, vec_cnt);
}

SEE ALSO

wsio_intr_assign_cpus(9F), wsio_intr_req_loc(9F)
NAME

wsio_intr_get_txn_info - Get the transaction address and data value.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_get_txn_info(
    struct isc_table_type *isc,
    wsio_intr_object_t obj,
    intptr_t *txn_addr,
    intptr_t *txn_data
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
obj Interrupt object.
txn_addr Transaction address value.
txn_data Transaction data value.

DESCRIPTION

The wsio_intr_get_txn_info WSIO function returns the transaction address and transaction data associated with a transaction based obj.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_INTR_INV_OBJ Must be a transaction-based obj; call wsio_intr_set_cpu_spec first.
WSIO_ERROR Failed to get values.
WSIO_PARM_ERROR Invalid parameters.

CONSTRAINTS

None

EXAMPLE

/* Allocate a transaction based interrupt */
wsio_intr_object_t obj;
int status;
intptr_t txn_addr, txn_data;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
    (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a TXN based interrupt */
status = wsio_intr_set_cpu_spec(isc, obj, WSIOINTR_CPU_ANY);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {

return(ERROR);
}

/* get the TXN values */
status = wsio_intr_get_txn_info(isc, obj,
   &txn_addr, &txn_data);
if (status != WSIO_OK) {
   return(ERROR);
}
printf("txn_addr = %ld\n", txn_addr);
printf("txn_data = %ld\n", txn_data);

SEE ALSO

wsio_intr_activate(9F), wsio_intr_alloc(9F), wsio_intr_set_cpu_spec(9F)
NAME

wsio_intr_req_loc - Retrieve interrupt vector assignments.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_req_loc(
    struct isc_table_type *isc,
    int cnt,
    uint32_t hints
);

PARAMETERS

isc Pointer to the isc structure of the interface card.

cnt The number of vector entries requested.

hints Selection criteria for the interrupt locality vectors.

WSIO_INTR_CARD_LOC Use the locality of the PCI card.

WSIO_INTR_ALL_LOC Use all available localities in the system.

WSIO_DRV_HIGH_PERF Indicates this is a high performance driver.

DESCRIPTION

Call the wsio_intr_req_loc service to request cnt locality interrupt resource vectors for the interface represented by the isc structure. Use the hints to indicate the preference for locality selection. You can use the inclusive OR operation to create the following combinations:

- (WSIO_INTR_CARD_LOC | WSIO_DRV_HIGH_PERF)
- (WSIO_INTR_ALL_LOC | WSIO_DRV_HIGH_PERF)

The (WSIO_INTR_CARD_LOC | WSIO_INTR_ALL_LOC) combination is not valid.

RETURN VALUES

WSIO_OK The operation was successful.

WSIO_ERROR Another error occurred.

EXAMPLES

#define MAX_VECTORS 64
my_msi_driver_init()
{
    intr_loc_t *my_loc;
    spu_t cpu_list[MAX_VECTORS];
    wsio_msi_set_info_t msi_args[MAX_VECTORS];
    int vec_cnt = MAX_VECTORS;

    if (wsio_msi_capability(isc, WSIO_MSI_X_CAPABILITY, vec_cnt) != WSIO_OK)
        return ERROR;

    if ((wsio_intr_req_loc(my_isc, vec_cnt, WSIO_INTR_ALL_LOC)) != WSIO_OK)
        return ERROR;

    if ((wsio_intr_get_loc(my_isc, &vec_cnt, &my_loc)) != WSIO_OK)
        return ERROR;

    if ((wsio_intr_assign_cpus( my_isc, vec_cnt, my_loc, cpu_list )) != WSIO_OK)
        return(ERROR);
if (wsio_msi_alloc(isc,WSIO_MSI_X_TYPE,vec_cnt,0,&msi_obj) != WSIO_OK)
    return ERROR);

for (i = 0; i < vec_cnt; i++)
{
    msi_args[i]->mask = WSIO_MSI_SET_CPU | WSIO_MSI_SET_ISR;
    msi_args[i]-> isr  = my_drv_isr;
    msi_args[i]->isr_arg = isc;
    msi_args[i]->cpu_id = my_intr_vec->cpu;
}

msi_hints = WSIO_MSI_ASSIGN_RANGE;
if(wsio_msi_assign(msi_obj,msi_hints,msi_args,0,vec_cnt)) == WSIO_OK)
{
    msi_hints = WSIO_MSI_VECTOR_RANGE;
    ret =  wsio_msi_enable(msi_obj, msi_hints, 0, vec_cnt);
}

SEE ALSO

wsio_intr_assign_cpus(9F), wsio_intr_get_loc(9F)
NAME

wsio_intr_set_attribute - Set the attribute value for an interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_set_attribute(
    IN struct isc_table_type *isc,
    IN wsio_intr_object_t *iobj,
    IN wsio_intr_attrib_t attrib,
    OUT intptr_t attr_val
);

PARAMETERS

isc Pointer to the device ISC structure.
iobj Interrupt object returned by the wsio_intr_alloc call.
attrib The interrupt attribute to change.
attr_val Value to be assigned to the attribute.

DESCRIPTION

This service is called to set an attribute value associated with the interrupt object. The interrupt corresponding to iobj must be disabled before making this call. This can be accomplished by calling the wsio_intr_deactivate or wsio_intr_deactivate_nowait call. Refer to the respective manpages for further information.

This routine can be called from a non-blocking context.

Currently there are no attributes that can be set using this call.

RETURN VALUES

WSIO_OK The attribute was successfully changed.
WSIO_INVALID_ISC Not a valid ISC structure.
WSIO_INTR_INV_OBJ Not a valid interrupt object.
WSIO_INTR_ACTIVATED The interrupt object is activated, so an attribute can not be sent.
WSIO_NOT_IMPLEMENTED The specified interrupt attribute is not implemented.
WSIO_PARM_ERROR The one or more arguments passed to the wsio_intr_set_attribute call were invalid.
WSIO_ERROR Setting the attribute failed.

CONSTRAINTS

This service must be called on an interrupt object that has been disabled. See the wsio_intr_deactivate(9F) or wsio_intr_deactivate_nowait(9F) for more information.

EXAMPLES

wsio_intr_object_t iobj;
wsio_intr_attrib_t attrib;
intptr_t attr_val;

status = wsio_intr_set_attribute (isc, iobj, attrib, attr_val);
SEE ALSO

wsio_intr_alloc(9F), wsio_intr_free(9F), wsio_intr_activate(9F), wsio_intr_deactivate(9F),
wsio_intr_deactivate_nowait(9F), wsio_intr_set_cpu_spec(9F), wsio_intr_set_irq_line(9F),
wsio_intr_get_attribute(9F)
NAME

wsio_intr_set_cpu_spec - Initialize and distribute transaction based interrupts.

SYNOPSIS

#include <sys/wsio.h>
(
    struct isc_table_type *isc,
    wsio_intr_object_t obj,
    intptr_t cpu_spec
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
obj Interrupt object.
cpu_spec CPU load balancing specification. If cpu_spec is neither of the following flags but is a txn_addr, the interrupt is bound to this CPU.

- WSIO_INTR_CPU_ANY The services select any processor. The device driver does not care which processor it is bound to.
- WSIO_INTR_CPU_ANY_UNIQUE The services do not select the same processor for an obj (only one obj per processor for this isc).
- txn_addr Bind to this CPU address.

DESCRIPTION

The wsio_intr_set_cpu_spec WSIO function is used to initialize and distribute transaction based interrupts.

RETURN VALUES

- WSIO_OK Successful completion.
- WSIO_INTR_INV_OBJ Must be a transaction based obj; call wsio_intr_set_cpu_spec first.
- WSIO_ERROR Failed to set cpu_spec.
- WSIO_EXCLUSIVE_FAILED Can not get an exclusive interrupt.
- WSIO_INTR_ACTIVATED obj is currently active; must call wsio_intr_deactivate first.
- WSIO_NOT_IMPLEMENTED cpu_spec algorithm not implemented.
- WSIO_INTR_INV_CPU_NUM Invalid CPU number specified.
- WSIO_PARM_ERROR Invalid parameters.

CONSTRAINTS

Do not call from the driver_attach routine.
Can be called from the driver_if_init routine.

EXAMPLE

/* Allocate a transaction based interrupt */
wsio_intr_object_t obj;
int status;
intptr_t txn_addr, txn_data;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
            (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a TXN based interrupt */
status = wsio_intr_set_cpu_spec(isc, obj, WSIO_INTR_CPU_ANY);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* get the TXN values */
status = wsio_intr_get_txn_info(isc, obj,
            &txn_addr, &txn_data);
if (status != WSIO_OK) {
    return(ERROR);
}

printf ("txn_addr = %ld\n", txn_addr);
printf ("txn_data = %ld\n", txn_data);

SEE ALSO

wsio_intr_activate(9F), wsio_intr_alloc(9F), wsio_intr_set_cpu_spec(9F)
NAME

wsio_intr_set_irq_line - Set the interrupt line number.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_set_irq_line(
    struct isc_table_type *isc,
    wsio_intr_object_t obj,
    intptr_t *irq_line_num,
    uint64_t flags
);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
obj Interrupt object.
irq_line_num The interrupt line number, or WSIO_IO_IRQ_LINE_AUTO
flags Zero (level sensitive) or WSIO_INTR_EDGE_SENSITIVE

DESCRIPTION

The wsio_intr_set_irq_line WSIO function is used to setup a line based obj. The
WSIO_INTR_ACTIVATED error code will be returned if this function is called on an active
interrupt object. Most drivers will use WSIO_IRQ_LINE_AUTO for the value of the irq_line_num
parameter which forces the services to determine the interrupt line value for the particular device.
If the device generates level-sensitive interrupts, the flags parameter must be zero. If the device
generates edge-sensitive interrupts, the flags parameter must be WSIO_INTR_EDGE_SENSITIVE.

RETURN VALUES

WSIO_OK Operation succeeded.
WSIO_INTR_INV_OBJ Must not be a transaction based interrupt.
WSIO_INTR_ACTIVATED Object is active; do not call wsio_intr_activate first.
WSIO_ERROR Failed to set line number.
WSIO_INTR_INV_FLAG Must be zero (level) or WSIO_INTR_EDGE_SENSITIVE.
WSIO_PARM_ERROR Invalid parameters.

CONSTRAINTS

Do not call from the driver_attach routine.
Can be called from the driver_if_init routine.

EXAMPLE

/* Allocate a line based interrupt */
wsio_intr_object_t obj;
int status;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
    (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}
/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
       WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
    return(ERROR);
}

**SEE ALSO**

wsio_intr_activate(9F), wsio_intr_alloc(9F), wsio_intr_set_line(9F)
NAME

wsio_io_sync - Perform a sync of shared memory if necessary.

SYNOPSIS

#include <sys/wsio.h>

void wsio_io_sync(

    struct isc_table_type *isc
);

PARAMETERS

isc  Pointer to the driver's isc_table entry.

DESCRIPTION

The wsio_io_sync WSIO function synchronizes a device's view and the host's view of memory. This functionality can also be achieved by doing a PIO read of a card register, but in some implementations this function may provide a lower latency mechanism. The wsio_io_sync function can be called in a non-blocking context.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

wsio_io_sync(isc_entry);

SEE ALSO

WSIO_ORDERED_INTERRUPTS(9F)
NAME

wsio_iova_to_phys - Translate an I/O virtual address to a physical address.

SYNOPSIS

#include <sys/wsio.h>

caddr_t wsio_iova_to_phys(
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_iova_t iova
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
dma_handle DMA handle allocated using wsio_allocate_dma_handle.
iova I/O virtual address to be translated.

DESCRIPTION

The wsio_iova_to_phys WSIO function is called by a device driver to translate an I/O virtual address to a physical address.
The wsio_iova_to_phys function can be called in a non-blocking context.

RETURN VALUES

The physical address corresponding to iova or NULL if a translation does not exist.

CONSTRAINTS

None

EXAMPLE

caddr_t phys_address = NULL;

phys_address = wsio_iova_to_phys(isc_entry,dma_handle,io_address);

/* The 'phys_address' variable will now contain the physical
 * address that 'iova' translates to.
 */

SEE ALSO

wsio_allocate_dma_handle(9F), wsio_allocate_shared_mem(9F), wsio_dma_pass_thru(9F),
wsio_fastmap_dma_buffer(9F), wsio_free_dma_handle(9F), wsio_free_shared_mem(9F),
wsio_flush_shared_mem(9F), wsio_init_map_context(9F), wsio_map_dma_buffer(9F),
wsio_remap_dma_buffer(9F), wsio_set_device_attributes(9F), wsio_unmap_dma_buffer(9F)
NAME

wsio_isc_to_hwpath - Returns the hardware path of the HBA.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_isc_to_hwpath(
    IN struct isc_table_type *isc,
    OUT char *path
);

PARAMETERS

isc IN: The isc handle of the interface.

path IN: A pointer to a string.

OUT: The string initialized with the hardware path.

DESCRIPTION

This service returns the hardware path of an interface described by the isc parameter. The
hardware path is returned in the path parameter.

RETURN VALUES

WSIO_OK Path returned successfully.

WSIO_ERROR An error occurred.

CONSTRAINTS

None

SEE ALSO

wsio_hwpath_to_isc(9F)
wsio_isc_to_instance(9F)

NAME

wsio_isc_to_instance - Retrieve an instance number of an iotree node.

SYNOPSIS

#include <sys/wsio.h>
#include <sys/ioparams.h>

int wsio_isc_to_instance(
    struct isc_table_type *isc,
    hw_path_t *dev_hw_path
);

PARAMETERS

isc A pointer to the ISC structure associated with the interface card for the device.
dev_hw_path A pointer to a structure containing device hardware path information relative to the interface card, or NULL if the card instance is desired.

DESCRIPTION

The wsio_isc_to_instance WSIO function provides an instance number of an iotree node that is a descendant of the card or device node specified by the isc and dev_hw_path parameters, according to the following rules:

- If isc is valid and dev_hw_path is NULL, the instance number of an iotree node corresponding to the interface card is returned.
- If isc is valid and dev_hw_path contains a path to a valid device (relative to the device's interface card), the instance number of the iotree node corresponding to the device is returned. SCSI interface drivers must check the instance number returned by wsio_isc_to_instance. If the returned value is greater than SCSI_MAX_BUS_ID, the driver must return WSIO_ERROR.

This function will not provide valid instance numbers if it is accessed before the driver has actually claimed a device (that is, before the driver's attach routine has successfully claimed and initialized an ISC structure). Until that point, the driver is not associated with an iotree node and will not have a valid instance number. Using this service within a driver's driver_if_init or driver_dev_init routines and later will yield a valid result.

RETURN VALUES

>=0 Successful completion. The value is the matching instance number.
-1 Error.

CONSTRAINTS

None
NAME
wsio_legacy_info_t - WSIO legacy node registration structure.

SYNOPSIS
#include <sys/wsio.h>

DESCRIPTION
The wsio_legacy_info_t structure, defined in <sys/wsio.h>, is used to register a legacy I/O node. It is passed as a parameter to wsio_reg_legacy.

STRUCTURE MEMBERS
typedef struct wsio_legacy_info {
    int version;
    hw_path_t hw_path;
    int type;
    wsio_drv_info_t *drv_info;
    int instance;
    char *name;
    char *desc;
    char id[16];
    char *bus_type;
    void *node
} wsio_legacy_info_t;

Field Description
version Data structure version number.
hw_path Pointer to a hardware path structure.
type Type of legacy I/O node.
drv_info A pointer to the drv_info_t structure.
instance An assigned class instance number.
name Optional name for name property.
desc Description property for the node.
id Id property for the node.
bus Bus type for the I/O node.
node The legacy I/O node.

SEE ALSO
wsio_async_scan(9F), wsio_claim_node(9F), wsio_destroy_legacy(9F), wsio_get_drv_priv(9F),
wsio_get_legacy(9F), wsio_node_get_isc(9F), wsio_put_drv_priv(9F), wsio_reg_legacy(9F),
wsio_reg_node(9F), wsio_reg_node_t(9S)
WSIO_LITTLE_ENDIAN(9F)

NAME

WSIO_LITTLE_ENDIAN - Macro to return true (1) if the local bus is little-endian.

SYNOPSIS

#include <sys/wsio.h>

int WSIO_LITTLE_ENDIAN(
    struct isc_table_type *isc
);

PARAMETERS

isc Pointer to the driver's isc_table entry.

DESCRIPTION

The WSIO_LITTLE_ENDIAN macro is called by a device driver to report whether
the local bus is little-endian. If it is, true is returned; otherwise it returns false. This can be used by a
driver along with the known endianness of the host processor to decide whether
endián swapping must be performed. Endian swapping might be necessary for any
data transfers between the I/O bus and local host memory.

WSIO_LITTLE_ENDIAN can be called in a non-blocking context.

RETURN VALUES

Returns a one (1) if the local bus is little-endian, and a zero (0) otherwise.

CONSTRAINTS

None

EXAMPLE

if (WSIO_LITTLE_ENDIAN(isc_entry)) {
    /* Endian swapping must be performed
} else {
    /* No endian swapping necessary */
}

SEE ALSO

WSIO_BIG_ENDIAN(9F)
NAME

wsio_map_cfg_handle - Obtain a configuration space access handle.

SYNOPSIS

#include <sys/wsio.h>

int wsio_map_cfg_handle(
    struct isc_table_type *isc,
    wsio_addr_handle_t *cfg_handle
);

PARAMETERS

isc Pointer to the driver's isc_table entry.

cfg_handle Pointer to contain the configuration handle upon completion.

DESCRIPTION

The wsio_map_cfg_handle WSIO function is called by device drivers to obtain a handle to access configuration space. wsio_map_cfg_handle must not be called in a non-blocking context.

RETURN VALUES

WSIO_OK Indicates a handle was successfully returned in cfg_handle.

WSIO_ERROR Indicates there was an error obtaining a handle.

CONSTRAINTS

Do not call in an interrupt context.

EXAMPLE

wsio_addr_handle_t handle;

if (wsio_map_cfg_handle(isc_entry,&handle) != WSIO_OK) {
    /* Error obtaining configuration space handle. Return an error code */
    return(ERROR);
}

/* The configuration space handle is now in the handle */

SEE ALSO

wsio_cfg_inXX(9F), wsio_cfg_outXX(9F), wsio_unmap_cfg_handle(9F)
NAME

wsio_map_dma_buffer - Map an existing memory object for packet DMA.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_map_dma_buffer(
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_map_context_t *context,
    wsio_dma_buffer_hints_t hints,
    wsio_range_type_t range_type,
    wsio_dma_map_t *host_range,
    wsio_dma_map_t *io_range
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
dma_handle DMA handle allocated using wsio.allocate_dma_handle.
context Pointer to the context used for mapping.
hints Bitmask that provides mapping hints. The allowable hints are as follows:

<table>
<thead>
<tr>
<th>hints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_DMASAFE</td>
<td>Forces coherent transactions to be used even for full-cacheline transactions. In some implementations semi-coherent transactions are used to enhance performance when it is known that the entire cacheline will be modified. Affects inbound DMA only.</td>
</tr>
<tr>
<td>WSIO_DMALOCK</td>
<td>In some implementations this allows atomic access to memory for devices using bus-lock primitives.</td>
</tr>
<tr>
<td>WSIO_DMA_FLUSH_ON_USE</td>
<td>In some implementations this hint tells the hardware to flush resources associated with this buffer after they are used. This inhibits coalescing transactions into larger transfers for cases where it is not beneficial.</td>
</tr>
<tr>
<td>WSIO_DMAIGN_ALIGNMENT</td>
<td>Indicates wsio_map_dma_buffer must not automatically set the IO_SAFE hint for partial cacheline buffers.</td>
</tr>
<tr>
<td>WSIO_DMA_CONTIGUOUS</td>
<td>Indicates wsio_map_dma_buffer must allocate a single contiguous IOVA. If wsio_map_dma_buffer is unable to do this, it returns WSIO_MAP_E_PARAMETER_ERROR. This hint implies WSIO_IO_IGN_ALIGNMENT.</td>
</tr>
<tr>
<td>WSIO_DMA_NO_SEQ</td>
<td>Inhibits hardware prefetching for outbound DMA in some implementations.</td>
</tr>
<tr>
<td>WSIO_DMA_INBOUND</td>
<td>Indicates the buffer is used exclusively for inbound DMA.</td>
</tr>
</tbody>
</table>
WSIO_DMA_OUTBOUND Indicates the buffer is used exclusively for outbound DMA.
WSIO_DMA_NULL Forces all hint values to zero. A hint value of zero tells the BN-CDIO to take hint values from the DMA object.

range_type Indicates the type of host memory being mapped. It can be:
- KERNELSPACE Indicates host_range is a kernel virtual buffer.
- PHYSICAL Indicates host_range is a physical buffer.
- > 0 Indicates host_range is in user space, and this is the space ID of the virtual address.

host_range Pointer to an address/length structure that contains information about the host space to map. If the mapping was only partially completed, this contains information about the remaining space to be mapped when the call completes.

io_range Pointer to an address or length structure containing information about the I/O space that was mapped.

DESCRIPTION

The wsio_map_dma_buffer WSIO function is called by a device driver to map an existing memory object for packet DMA. If continuous DMA is required wsio_allocate_shared_mem must be used. Continuous DMA must be used for control structures, circular buffers, or any kind of buffer that needs to be accessed on a continuous basis by the I/O device. Packet DMA must be used when the mappings are temporary, or when pre-existing memory objects must be mapped for DMA.

If the entire host range cannot be mapped, the buffer may only be partially mapped. This will be indicated by a return value of WSIO_MAP_W_PARTIAL. In this case, wsio_map_dma_buffer will need to be called again to map more of the host range. Using this method of calling wsio_map_dma_buffer multiple times, the entire host range can be mapped into multiple I/O virtual ranges.

All mappings remain in effect until wsio_unmap_dma_buffer or wsio_remap_dma_buffer are called to remove or change them.

Callers are guaranteed buffers of at least 4K Bytes can be mapped into a contiguous range of I/O virtual address.

The context parameter is used to insure I/O resources are used efficiently. This structure must be initialized via a call to wsio_init_map_context. The same context must be used for any group of mappings that will all be unmapped at the same time. If multiple buffers will not be mapped into one context, NULL can be passed in instead of a valid context.

Device drivers can set up a callback routine that will come into play if resources are not available at the time a mapping is attempted. If this callback is set up, WSIO_MAP_W_CALLBACK will be returned instead of a no resource error. When resources become available, the callback routine will be called to indicate this to the device driver. For more information on how to setup and use a callback, see the wsio_set_dma_callback manpage.

wsio_map_dma_buffer can be called in a non-blocking context.

RETURN VALUES

- WSIO_MAP_OK Returned if the entire buffer has been mapped.
- WSIO_MAP_W_PARTIAL Returned if only part of the buffer has been mapped.
- WSIO_MAP_W_CALLBACK Returned if no resources are available and a callback function exists.
WSIO_MAP_E_NO_RESOURCES  Returned if no resources are available and no callback function exists.

WSIO_MAP_E_RESOURCE_ERROR Returned if the request cannot and will never succeed.

WSIO_MAP_E_HIGH_ADDR  Returned if the call failed because the device cannot reach the destination address.

WSIO_MAP_E_PARAMETER_ERROR Returned if an invalid parameter has caused failure of the call.

WSIO_MAP_E_UNKNOWN_ERROR Returned for hardware or other errors.

**CONSTRAINTS**
None

**EXAMPLE**

```c
void *dma_handle;
wsio_map_context_t dma_con;
wsio_dma_map_t host_range, io_range;

dma_handle = wsio_allocate_dma_handle(isc_entry);
wsio_init_map_context(&dma_con);

host_range.iov_base = host_virtual_address;
host_range.iov_len = dma_buffer_length;

if (wsio_map_dma_buffer(isc_entry,dma_handle,dma_con,
    WSIO_DMA_OUTBOUND,KERNELSPACE,&host_range,&io_range
    )!= WSIO_MAP_OK) {
    /* Unable to perform the mapping so return an error */
    return(ERROR);
}

/* The host virtual buffer represented in the above
 * code by host_virtual_address' is now mapped. Note that
 * this code does not handle the case where the buffer is
 * only partially mapped. In that case, a more
 * complete example would call wsio_map_dma_buffer() again
 * each time WSIO_MAP_PARTIAL was returned and save each
 * io_range that was returned.
 */
```

**SEE ALSO**

wsio_allocate_dma_handle(9F), wsio_allocate_shared_mem(9F), wsio_dma_pass_thru(9F),
wsio_fastmap_dma_buffer(9F), wsio_free_dma_handle(9F), wsio_free_shared_mem(9F),
wsio_flush_shared_mem(9F), wsio_iova_to_phys(9F), wsio_init_map_context(9F),
wsio_remap_dma_buffer(9F), wsio_set_device_attributes(9F), wsio_set_dma_attributes(9F),
wsio_unmap_dma_buffer(9F)
NAME

wsio_map_port - Obtain an I/O port handle.

SYNOPSIS

#include <sys/wsio.h>

int wsio_map_port(
    struct isc_table_type *isc,
    int32_t port_addr,
    sizet_t size,
    wsio_addr_handle_t port_handle
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
port_addr Address of the port obtained from wsio_get_ioports.
size Size of the port to be mapped.
port_handle The port handle upon completion.

DESCRIPTION

The wsio_map_port WSIO function will attempt to map an I/O port. After a port has been
mapped, it can be read from and written to using the wsio_port_inXX and wsio_port_outXX
accessor functions. Do not call the wsio_map_port function in a non-blocking context.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_ERROR There was a parameter error.

CONSTRAINTS

Do not call in an interrupt context.

EXAMPLE

wsio_addr_handle_t port_handle;
wsio_iop_t ioports_array[10];
/* An array with enough space for all ports needs to be
 * allocated */

if (wsio_get_ioports(isc_entry,10,ioports_array) != WSIO_OK) {
    /* There was a problem obtaining the ports */
    return(ERROR);
}

if (wsio_map_port(isc_entry,ioports_array[0].addr,
    ioports_array[0].size, &port_handle) != WSIO_OK) {
    /* There was an error mapping the port */
    return(ERROR);
}

/* The first I/O port will now be mapped and can be
 * accessed via 'port_handle' */

SEE ALSO

wsio_get_ioports(9F), wsio_port_inXX(9F), wsio_port_outXX(9F), wsio_unmap_port(9F)
NAME

wsio_map_reg - Map device registers to host memory space.

SYNOPSIS

#include <sys/wsio.h>

int wsio_map_reg(
    struct isc_table_type *isc,
    wsio_reg_info_t *reg_info
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
reg_info Pointer to information about the register to be mapped. This structure must be obtained from wsio_get_all_registers. It will also be used after the mapping to access the register.

DESCRIPTION

The wsio_map_reg WSIO function, given information about a register to be mapped, will attempt to map a register so that it is accessible via the wsio_read_regXX and the wsio_write_regXX functions. After the mapping has been performed, the reg_info variable can be used to access the register. Prior to the mapping, reg_info must be obtained from a call to wsio_get_all_registers. Do not call the wsio_map_reg function in a non-blocking context.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_ERROR Could not create the mapping.

CONSTRAINTS

Do not call in an interrupt context.

EXAMPLE

wsio_reg_info_t *registers;

registers = wsio_get_all_registers(isc_entry);
if (registers == NULL) {
    /* No registers exist. Return an error */
    return(ERROR);
}

if (wsio_map_reg(isc_entry,&registers[1]) != WSIO_OK) {
    /* The second device register (index 1 into the array) will now be mapped. */
    return(ERROR);
}

SEE ALSO

wsio_get_all_registers(9F), wsio_read_regXX(9F), wsio_write_regXX(9F), wsio_unmap_reg(9F)
NAME

wsio_msi_alloc - Allocate a Message-Signaled Interrupt object.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_msi_alloc(
    struct isc_table_type *isc,
    wsio_msi_type_t type,
    int vector_cnt,
    wsio_msi_hints_t hints,
    wsio_msi_hndl_t *hndl
);

PARAMETERS

isc Pointer to the driver's isc_table entry.

type Type of interrupt object to allocate, either WSIO_MSI_TYPE or
      WSIO_MSI_X_TYPE.

vector_cnt Number of interrupt vectors requested.

hints Reserved for future use.

hndl Returned interrupt object.

DESCRIPTION

The wsio_msi_alloc service is called to allocate an MSI/MSI-X interrupt object.
Valid values for type are shown in the following table:

Table 2-6 Values for type

<table>
<thead>
<tr>
<th>wsio_msi_type_t</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_MSI_TYPE</td>
<td>Allocate an interrupt object for MSI type interrupts.</td>
</tr>
<tr>
<td>WSIO_MSI_X_TYPE</td>
<td>Allocate an interrupt object for MSI-X type interrupts.</td>
</tr>
</tbody>
</table>

The value passed in for vector_cnt must not exceed the number of vectors supported by the
MSI/MSI-X function. The latter information can be obtained using the wsio_msi_capability
service. Drivers must always call wsio_msi_capability to determine the hardware and
platform capabilities or limitations before calling this service. For an object of WSIO_MSI_TYPE,
vector_cnt must be equal to one. The service will return WSIO_ERROR if it is not.

The hints field is reserved for future hints. It must be set to zero.

If the call is successful the handle for the new MSI object is returned in the hndl parameter.

A PCI card cannot enable both MSI and MSI-X functions at the same time. If wsio_msi_alloc
is called to allocate an interrupt object of one type, a subsequent call to wsio_msi_alloc to
allocate an interrupt object of the other type will result in an error.

Any driver using MSI or MSI-X interrupts must register for the WSIO_EVENT_OFFLINE_CPU
and WSIO_EVENT_MSI_INTR_MIGR events. If the isc has not registered for both of those events,
wsio_msi_alloc returns WSIO_ERROR.

RETURN VALUES

WSIO_OK Successful completion.
**WSIO_ERROR**
A MSI/MSI-X interrupt object has already been allocated for this PCI function, or an error occurred.

**WSIO_INVALID_ISC**
The isc is not valid.

**WSIO_INVALID_PARM**
The type or hints parameter is invalid, the PCI function does not support the given type, or vector_cnt is an invalid value.

**WSIO_NOT_SUPPORTED**
The platform does not have support for MSI/MSI-X features.

**CONSTRANTS**
Do not call while holding a spinlock or while executing in interrupt context.

**EXAMPLE**
```c
wsio_ret_code_t ret;
int vector_cnt;
wsio_msi_hndl_t msi_hndl;

/* Allocate a maximum of 5 interrupt vectors */
ret = wsio_msi_capability(isc, WSIO_MSI_X_CAPABILITY, &vector_cnt);if (vector_cnt > 0) {
    if (vector_cnt > 5) vector_cnt = 5;
    ret = wsio_msi_alloc(isc, WSIO_MSI_X_TYPE, vector_cnt, 0, &msi_hndl);}
```

**SEE ALSO**
wsio_msi_assign(9F), wsio_msi_capability(9F), wsio_msi_disable(9F), wsio_msi_enable(9F),
wsio_msi_free(9F), wsio_msi_get_cpus(9F), wsio_msi_query(9F), wsio_msi_resize(9F)
NAME

wsio_msi_assign - Initialize Message-Signaled Interrupt vectors.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_msi_assign(
    wsio_msi_hndl_t hndl,
    wsio_msi_assign_type_t assign_type,
    wsio_msi_setinfo_t *vec_info,
    int qualifier,
    size_t num
);

PARAMETERS

hndl Handle of an MSI/MSI-X interrupt object.
assign_type How to set the per-vector information.
vec_info Array of wsio_msi_setinfo_t structures, describing the vector values.
qualifier The starting vector, or a CPU identifier. Its meaning depends on the assign_type.
um The number of vectors to modify, or the number of elements in vec_info. Its meaning depends on the assign_type.

DESCRIPTION

The wsio_msi_assign service initializes a set of interrupt vectors belonging to an MSI/MSI-X interrupt object. All vectors being modified must be disabled, otherwise wsio_msi_assign will return an error.

Values for assign_type are shown in the table below.

Table 2-7 Values for assign_type

<table>
<thead>
<tr>
<th>wsio_msi_assign_type_t</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_MSI_ASSIGN_RANGE</td>
<td>Indicates that a contiguous range of vectors must be modified. The parameters qualifier indicates the start of the range and num indicates the size.</td>
</tr>
<tr>
<td>WSIO_MSI_CPU_ID</td>
<td>Indicates that the parameter qualifier is a CPU identifier. Only those vectors assigned to the specified CPU will be modified.</td>
</tr>
<tr>
<td>WSIO_MSI_ASSIGN_PER_VEC</td>
<td>Indicates that the vec_info array contains an individual entry for each vector to be modified. This flag can be OR'd together with either WSIO_MSI_ASSIGN_RANGE or WSIO_MSI_CPU_ID.</td>
</tr>
<tr>
<td>WSIO_MSI_ASSIGN_ALL</td>
<td>Indicates that the vec_info array contains a single entry that must be applied to all the specified vectors. This flag can be OR'd together with either WSIO_MSI_ASSIGN_RANGE or WSIO_MSI_CPU_ID.</td>
</tr>
</tbody>
</table>

Valid flag combinations for assign_type include:

(WSIO_MSI_ASSIGN_RANGE | WSIO_MSI_ASSIGN_PER_VEC) Assign a range of contiguous vectors and the vector array has individual assignments for each. There is a 1 to 1 correspondence between the entries in the vec_info array and the number of vectors to modify hence the parameter num indicates the number of entries in the vec_info array and the number of vectors to modify starting from the vector indicated by qualifier.
Assign a range of contiguous vectors. The vector array has a single assignment for all. The `vec_info` array contains a single entry and `num` indicates the number of vectors to modify starting from the vector indicated by `qualifier`.

Assign all vectors that are currently assigned to the CPU specified in `qualifier`. The vector array has individual assignments for each vector hence `num` indicates the total number of entries in the `vec_info` array.

Assign all vectors that are currently assigned to the CPU specified in `qualifier`. The vector array has a single assignment for all. The value of `num` is ignored.

The `vec_info` parameter points to an array of structures of type `wsio_msi_setinfo_t`. Its definition is as follows:

```c
typedef struct wsio_msi_setinfo {
    int       mask;
    int       (isr*)();
    void     *isr_arg;
    wsio_cpu_id_t cpu_id;
} wsio_msi_setinfo_t;
```

- `mask` The bit-wise OR of flag values that describe what other fields are valid, taken from:  
  - `WSIO_MSI_SET_CPU` The `cpu_id` field is valid.  
  - `WSIO_MSI_SET_ISR` The `isr` and `isr_arg` fields are valid.

- `isr` Pointer to the driver's interrupt service routine.

- `isr_arg` Driver defined parameter passed as the first parameter to `isr`. Typically, `isc` is passed as `isr_arg`.

- `cpu_id` CPU identifier. This is typically obtained from a call to `wsio_msi_get_cpus`.

For example, the caller could specify that all fields in the structure are valid by setting `mask` to `(WSIO_MSI_SET_CPU | WSIO_MSI_SET_ISR)` and then initializing all fields in the `wsio_msi_setinfo_t` structure.

Alternatively, the caller could specify that only the CPU identifier is to be set by setting `mask` to `WSIO_MSI_CPU_SET_ID`, and then only initializing the `cpu_id` in each `wsio_msi_setinfo_t` structure.

If multiple vectors of an MSI type interrupt object are being assigned then only the `cpu_id` in the first `wsio_msi_setinfo_t` structure will be used. This is because all of the MSI vectors are directly towards the same CPU. Only the data portion of the vector differs.

**RETURN VALUES**

- `WSIO_OK` Successful completion.
- `WSIO_ERROR` One or more vectors are active, or an error occurred.
- `WSIO_INVALID_PARM` Invalid value for a parameter was passed in.
- `WSIO_NOT_SUPPORTED` The platform does not have support for MSI/MSI-X features.

**CONSTRAINTS**

Do not call this service while holding a spinlock.

If CPU assignments are being modified, do not call this service while executing in interrupt context.

**EXAMPLES**

In the following example, the caller initializes 10 contiguous vectors of an MSI-X object starting at vector 5 with the same values.
wsio_msi_setinfo_t vec_vals;

/* Only 1 vector entry, so it is not an array */
vec_vals.mask    = WSIO_MSI_SET_CPU | WSIO_MSI_SET_ISR;
vec_vals.isr     = my_drv_isr;
vec_vals.isr_arg = my_isr_arg;
vec_vals.cpu_id  = cpu_id;

/* Vectors 5 thru 14 will be updated with the specified values */
wsio_msi_assign(hndl, (WSIO_MSI_ASSIGN_RANGE | WSIO_MSI_ASSIGN_ALL),
                 &vec_vals, 5, 10);

In the following example, the caller only wants to change the CPU assignment of all vectors
assigned to old_cpu_id.

wsio_msi_setinfo_t vec_vals;

/* Only the CPU id is changing, so set the mask appropriately */
vec_vals.mask    = WSIO_MSI_SET_CPU;
vec_vals.cpu_id  = new_cpu_id;/* Disable all vectors assigned to the old CPU id */
wsio_msi_disable(hndl, old_cpu_id, 0, 0); /* Change only the vectors associated with the old CPU */
type = (WSIO_MSI_CPU_ID | WSIO_MSI_ASSIGN_ALL);
wsio_msi_assign (hndl, type, &vec_vals, old_cpu_id, 0); /* Enable all the vectors assigned to the new CPU id */
wsio_msi_enable(hndl, new_cpu_id, 0, 0);

SEE ALSO
wsio_msi_alloc(9F), wsio_msi_capability(9F), wsio_msi_disable(9F), wsio_msi_enable(9F),
wsio_msi_free(9F), wsio_msi_get_cpus(9F), wsio_msi_query(9F), wsio_msi_resize(9F)
wsio_msi_capability(9F)

NAME

wsio_msi_capability - Report the MSI and MSI-X capabilities of a PCI interface card.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_msi_capability(  
    struct isc_table_type *isc,  
    wsio_msi_plat_cap_t capability,  
    int *value
);

PARAMETERS

capability    Type of capability to query.
isc            Pointer to the driver's isc_table entry.
value          Returned value indicating the MSI/MSI-X capability.

DESCRIPTION

The wsio_msi_capability service queries the MSI/MSI-X capabilities of a PCI card. The capabilities returned are determined by both the PCI card’s MSI/MSI-X hardware and any further limitations imposed by the platform.

Drivers must always call this service to determine the capabilities before calling wsio_msi_alloc to allocate an MSI or MSI-X object.

Valid values for capability are shown in the following table:

<table>
<thead>
<tr>
<th>wsio_msi_plat_cap_t</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_MSI_CAPABILITY</td>
<td>A returned value of 0 indicates that the PCI card does not support MSI. A value of 1, 2, 4, 8, 16 or 32 indicates the maximum number of vectors supported by the MSI function of the card.</td>
</tr>
<tr>
<td>WSIO_MSI_X_CAPABILITY</td>
<td>The service returns a number in the range of 0 to 2048 in the parameter value. A value of 0 indicates that MSI-X type interrupts are not supported; otherwise, a number between 1 to 2048, inclusive, indicates the maximum number of entries in the MSI-X vector table for this PCI device.</td>
</tr>
<tr>
<td>WSIO_MSI_MASK_CAPABILITY</td>
<td>The service returns a value indicating if the hardware supports per-vector masking for MSI and MSI-X. Valid results for value are: 0 Masking is not supported. 1 Masking is supported for MSI only. 2 Masking is supported for MSI-X only. 3 Masking is supported for both MSI and MSI-X.</td>
</tr>
</tbody>
</table>

The value returned for WSIO_MSI_CAPABILITY indicates if MSI type interrupts are supported and if so the maximum number of contiguous vectors that can be allocated. The platform determines this by first reading the Capability_ID register of the capabilities structure and if MSI is supported the Multiple_Message_Capable field of the Message_Control register of the MSI capabilities structure. The platform may impose further restrictions on the maximum number of vectors supported; in particular, the current MSI implementation on HP-UX only supports 1 vector.

The value returned for WSIO_MSI_X_CAPABILITY indicates if MSI-X type interrupts are supported and if so the size of the MSI-X table. The underlying platform code obtains this
information from the Capability_ID register of the capabilities structure and if MSI-X is supported the MSI-X_Table_Size field of the Message_Control register of the MSI capabilities structure. The value returned for WSIO_MSI_MASK_CAPABILITY indicates if per-vector masking is supported by the underlying hardware.

No hardware is initialized at this point.

**RETURN VALUES**

- WSIO_OK: Successful completion.
- WSIO_ERROR: An error occurred.
- WSIO_INVALID_ISC: The isc is not valid.
- WSIO_INVALID_PARM: The capability parameter contains invalid options, or the value parameter is NULL.
- WSIO_NOT_SUPPORTED: The platform does not have support for MSI/MSI-X features.

**CONSTRAINTS**

None

**EXAMPLE**

```c
int vector_cnt;

/* Determine if MSI-X type interrupts are supported */
wsio_msi_capability(my_isc, WSIO_MSI_X_CAPABILITY, &vector_cnt);
if (vector_cnt > 0) {
    printf("MSI-X table size == %d\n", vector_cnt);
}
else {
    /* MSI-X interrupts are not supported; are MSI type interrupts supported? */
    wsio_msi_capability(my_isc, WSIO_MSI_CAPABILITY, &vector_cnt);
    if (vector_cnt > 0) {
        printf("MSI interrupts vector count == %d\n", vector_cnt);
    } else {
        printf("Neither MSI nor MSI-X supported by card, using LBI\n");
    }
}
```

**SEE ALSO**

- `wsio_msi_alloc(9F)`, `wsio_msi_assign(9F)`, `wsio_msi_disable(9F)`, `wsio_msi_enable(9F)`, `wsio_msi_free(9F)`, `wsio_msi_get_cpus(9F)`, `wsio_msi_query(9F)`, `wsio_msi_resize(9F)`
wsio_msi_disable(9F)

NAME

wsio_msi_disable - Disable selected MSI/MSI-X interrupt vectors.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_msi_disable(
    wsio_msi_hndl_t hndl,
    int type,
    int first,
    size_t cnt
);

PARAMETERS

hndl Handle of an MSI/MSI-X interrupt object.

type Criterion to select the vectors to be disabled, either WSIO_MSI_VECTOR_RANGE, WSIO_MSI_ALL_VECTORS, or a valid CPU identifier.

first First vector to disable. This is only examined if type is WSIO_MSI_VECTOR_RANGE.

cnt Number of vectors to disable. This is only examined if type is WSIO_MSI_VECTOR_RANGE.

DESCRIPTION

The wsio_msi_disable service is called to disable individual MSI/MSI-X interrupt vectors. The parameter hndl is the MSI-X handle.

The parameter type indicates the criterion for selecting the vectors to disable:

• If type is WSIO_MSI_VECTOR_RANGE, first and cnt indicate a contiguous range of vectors to disable.

• If type is WSIO_MSI_ALL_VECTORS, all interrupts of the MSI function are disabled. This does not change any per-vector mask settings, and vectors that were disabled on a per-vector basis remain disabled when a call to wsio_msi_enable with type set to WSIO_MSI_ALL_VECTORS is made to re-enable the vectors.

• If type is set to a valid CPU identifier, only those MSI-X vectors that are programmed with the specified CPU will be disabled. CPU identifiers are distinct from the values of WSIO_MSI_VECTOR_RANGE and WSIO_MSI_ALL_VECTORS.

The MSI object must support vector masking, otherwise this service returns an error. Drivers can determine if the card supports per-vector masking by calling the service wsio_msi_capability with the capability parameter set to WSIO_MSI_MASK_CAPABILITY.

When called in process context, wsio_msi_disable waits for ISR completion. When it returns, the caller is guaranteed that no ISRs are currently executing. However, this guarantee does not apply when this service is called in ICS context or timeout context.

RETURN VALUES

WSIO_OK Successful completion.

WSIO_ERROR An error occurred.

WSIO_INVALID_PARM Invalid value for type, invalid CPU identifier, or invalid vector range specified by first and cnt parameters.

WSIO_NOT_SUPPORTED The platform does not have support for MSI/MSI-X features.
CONSTRAINTS
Do not call this service while holding a spinlock.

EXAMPLE

wsio_msi_hndl_t msi_hndl;

    /* Disable all vectors on the handle */
    wsio_msi_disable(msi_hndl, WSIO_MSI_ALL_VECTORS, 0, 0);

SEE ALSO

wsio_msi_alloc(9F), wsio_msi_assign(9F), wsio_msi_capability(9F), wsio_msi_enable(9F),
wsio_msi_free(9F), wsio_msi_get_cpus(9F), wsio_msi_query(9F), wsio_msi_resize(9F)
NAME

wsio_msi_enable - Enable selected MSI/MSI-X interrupt vectors.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_msi_enable(
    wsio_msi_hndl_t hndl,  
    int type,  
    int first,  
    size_t cnt  
) ;

PARAMETERS

hndl    Handle of an MSI/MSI-X interrupt object.

 type   Criterion to select the vectors to be enabled, either WSIO_MSI_VECTOR_RANGE, 
        WSIO_MSI_ALL_VECTORS, or a valid CPU identifier.

 first  First vector to enable. This is only examined if type is WSIO_MSI_VECTOR_RANGE.

 cnt    Number of vectors to enable. This is only examined if type is 
        WSIO_MSI_VECTOR_RANGE.

DESCRIPTION

The wsio_msi_enable service is called to enable MSI/MSI-X interrupt vectors. The parameter 
hndl is the MSI-X handle.

The parameter type indicates the criterion for selecting the vectors to enable:

• If type is WSIO_MSI_VECTOR_RANGE, first and cnt indicate a contiguous range of vectors to 
enable. This option is useful during PCI card initialization or when resizing the vector table.

• If type is WSIO_MSI_ALL_VECTORS, all interrupts of the MSI function and all vectors that 
are not masked off on a per-vector basis will be re-enabled.

• If type is set to a valid CPU identifier, only those MSI-X vectors that are programmed with 
the specified CPU will be enabled. This option is useful during interrupt migration or when 
new CPUs come online. Note that CPU identifiers are distinct from the values of 
WSIO_MSI_VECTOR_RANGE and WSIO_MSI_ALL_VECTORS.

For MSI type interrupts, if per-vector masking is not supported then this service will simply 
enable MSI type interrupts on the card. Also for MSI type interrupts, specifying a CPU identifier 
is not useful, as only a single CPU is assigned as the interrupt address.

RETURN VALUES

WSIO_OK        Successful completion.
WSIO_ERROR     An error occurred.
WSIO_INVALID_ISC The isc is not valid.
WSIO_INVALID_PARM Invalid value for type, invalid CPU identifier, or invalid vector range 
specified by first and cnt parameters.
WSIO_NOT_SUPPORTED The platform does not have support for MSI/MSI-X features.

CONSTRAINTS

Do not call this service while holding a spinlock.
There are restrictions on calling `wsio_msi_enable` in interrupt context. It must not be called in interrupt context if the vectors are being enabled for the first time after allocation or for the first time after the ISR or ISR argument have been modified. It may be called in interrupt context if the vectors have already been activated—that is, if the interrupt vector was previously enabled, then masked via `wsio_msi_disable`.

**EXAMPLE**

```c
wsio_msi_hndl_t msi_hndl;

/* Enable all the vectors on the handle */
wsio_msi_enable(msi_hndl, WSIO_MSI_ALL_VECTORS, 0, 0);
```

**SEE ALSO**

`wsio_msi_alloc(9F)`, `wsio_msi_assign(9F)`, `wsio_msi_capability(9F)`, `wsio_msi_disable(9F)`, `wsio_msi_free(9F)`, `wsio_msi_get_cpus(9F)`, `wsio_msi_query(9F)`, `wsio_msi_resize(9F)`
NAME

wsio_msi_free - Free a Message-Signaled Interrupt object.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_msi_free(
    wsio_msi_hndl_t *hndl
);

PARAMETERS

hndl  Previously allocated MSI/MSI-X interrupt object.

DESCRIPTION

The wsio_msi_free service deallocates an MSI-X interrupt object. All platform resources associated with the object are freed. The MSI hardware is disabled (by clearing the Enable bit in the MessageControl register of the MSI capabilities structure) thus assuring that no MSI or MSI-X interrupts can be generated.

When called in process context, wsio_msi_free waits for ISR completion. When it returns, the caller is guaranteed that no ISRs are currently executing. However, this guarantee does not apply when this service is called in ICS context or timeout context.

RETURN VALUES

WSIO_OK      Successful completion.
WSIO_ERROR   The hndl is not valid, or an error occurred.
WSIO_NOT_SUPPORTED The platform does not have support for MSI/MSI-X features.

CONSTRAINTS

Do not call this service while holding a spinlock or while executing in interrupt context.

EXAMPLE

wsio_ret_code_t ret;
wsio_msi_hndl_t msi_hndl;

/* Free the MSI interrupt object */
ret = wsio_msi_free(msi_hndl);

SEE ALSO

wsio_msi_alloc(9F), wsio_msi_assign(9F), wsio_msi_capability(9F), wsio_msi_disable(9F),
wsio_msi_enable(9F), wsio_msi_get_cpus(9F), wsio_msi_query(9F), wsio_msi_resize(9F)
**NAME**

wsio_msi_get_cpus - Get a list of CPUs available for I/O interrupts.

**SYNOPSIS**

```c
#include <sys/wsio.h>

wsio_ret_code_t wsio_msi_get_cpus(
    struct isc_table_type *isc,
    wsio_cpu_id_t *cpu_array,
    int *cnt,
    wsio_msi_cpu_hints_t hints
);
```

**PARAMETERS**

- **isc**  
  Pointer to the driver's isc_table entry.

- **cpu_array**  
  Array of CPU identifiers to be filled in by `wsio_msi_get_cpus`, or NULL.

- **cnt**  
  Number of elements in the `cpu_array`.

- **hints**  
  Selection criteria for CPUs.

**DESCRIPTION**

The `wsio_msi_get_cpus` service is called to get a list of CPUs that are available for I/O interrupts. The caller passes in an array of CPU identifiers in `cpu_array` and `wsio_msi_get_cpus` initializes the array with valid CPU identifiers. The `cnt` parameter indicates the size of the array when passed in and the number of valid entries on return.

Valid values for `hints` are shown in the following table:

<table>
<thead>
<tr>
<th><code>wsio_msi_cpu_hints_t</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_MSI_CPU_LOCAL</td>
<td>Return only CPUs local to the HBA, as described by the <code>isc</code> parameter.</td>
</tr>
<tr>
<td>WSIO_MSI_GET_ALL_CPUS</td>
<td>Return the identifiers of all platform CPUs that are available for I/O interrupts.</td>
</tr>
<tr>
<td>WSIO_MSI_NEXT_CPU</td>
<td>Return a single CPU identifier. If the caller passes in a CPU identifier in the <code>cpu_array</code>, the next CPU available, subject to the selection criteria, will be returned. If the caller passes in a CPU identifier of -1, the service will assign a CPU based on an internal round-robin algorithm. This hint is used mostly for MSI type interrupts to allow the platform code to round robin CPU assignments.</td>
</tr>
</tbody>
</table>

In case of the hints WSIO_MSI_CPU_LOCAL or WSIO_MSI_GET_ALL_CPUS, if the caller passes in a NULL value for the `cpu_array` parameter, `wsio_msi_get_cpus` will only update `cnt` to the number of CPUs that meet the selection criteria.

**RETURN VALUES**

- **WSIO_OK**  
  Successful completion. The identifiers of available CPUs are returned in the `cpu_array`.

- **WSIO_ERROR**  
  The system is not yet sufficiently initialized to provide a list of applicable CPUs, or an error occurred.

- **WSIO_INVALID_ISC**  
  The `isc` is not valid.

- **WSIO_INVALID_PARM**  
  The `cpu_array` is NULL (while `hints` is WSIO_MSI_NEXT_CPU), `cnt` is 0, or an invalid value for `hints` was specified.
### EXAMPLE

```c
#define MAX_CPUS 64 /* Assume that the maximum number of CPUs is 64 */

wsio_ret_code_t ret;
wsio_cpu_id_t cpu_ids[MAX_CPUS];
int array_size, active_cpus, i;

array_size = MAX_CPUS;
ret = wsio_msi_get_cpus(isc, cpu_ids, &array_size, WSIO_MSI_GET_ALL_CPUS);
active_cpus = array_size;

for (i = 0; i < active_cpus; i++) {
    printf("cpuid == %x\n", cpu_ids[i]);
}
```

### SEE ALSO

`wsio_msi_alloc(9F), wsio_msi_assign(9F), wsio_msi_capability(9F), wsio_msi_disable(9F), wsio_msi_enable(9F), wsio_msi_free(9F), wsio_msi_query(9F), wsio_msi_resize(9F)`
**wsio_msi_query(9F)**

**NAME**

*wsio_msi_query* - Return per-vector attributes of an MSI/MSI-X interrupt object.

**SYNOPSIS**

```c
#include <sys/wsio.h>

wsio_ret_code_t wsio_msi_query(
    wsio_msi_hndl_t *hndl,
    wsio_msi_vec_info_t type,
    int first,
    int num,
    void *vec_info
);
```

**PARAMETERS**

- `hndl` - Handle of an MSI/MSI-X interrupt object.
- `type` - Type of vector information to return.
- `first` - Starting vector.
- `num` - Number of vectors to query.
- `vec_info` - Pointer to a buffer large enough to hold the number of specified vector attribute values. Its data type will vary depending on the value of the `type` parameter, so it must be typecast to a void pointer.

**DESCRIPTION**

The `wsio_msi_query` service is called to query a contiguous set of vector values of an MSI or MSI-X interrupt object.

Valid values for `type`, as well as the data type returned in the `vec_info` array, are shown in the following table:

**Table 2-10 Values for type**

<table>
<thead>
<tr>
<th>wsio_msi_vec_info_t</th>
<th>type of vec_info</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_MSI_CPU_ID</td>
<td>wsio_cpu_id_t *</td>
<td>Return the identifier of the CPU that each vector's interrupt is directed to. The value returned in <code>vec_info</code> will be an array of type <code>wsio_cpu_id_t</code>.</td>
</tr>
<tr>
<td>WSIO_MSI_ENABLED</td>
<td>int *</td>
<td>Return the status of the vector mask. A 0 value indicates the vector is enabled, non-zero indicates that it is disabled. The value returned in <code>vec_info</code> will be an array of integers.</td>
</tr>
<tr>
<td>WSIO_MSI_OBJ_TYPE</td>
<td>wsio_msi_type_t *</td>
<td>Return the type of interrupt object, either <code>WSIO_MSI_TYPE</code> or <code>WSIO_MSI_X_TYPE</code>. The value returned in <code>vec_info</code> will be a single value of type <code>wsio_msi_type_t</code> since an interrupt object can only have one type.</td>
</tr>
<tr>
<td>WSIO_MSI_PENDING</td>
<td>int *</td>
<td>Return the status of the pending bit. A 0 value indicates an interrupt is pending, non-zero indicates that it is not pending. The value returned in <code>vec_info</code> will be an array of integers. If an MSI function does not support per-vector masking, <code>wsio_msi_query</code> will return <code>WSIO_NOT_SUPPORTED</code>.</td>
</tr>
<tr>
<td>WSIO_MSI_VECTOR_CNT</td>
<td>int *</td>
<td>Return the number of vectors associated with the interrupt object. The value returned in <code>vec_info</code> will be a single integer.</td>
</tr>
</tbody>
</table>
RETURN VALUES

WSIO_OK Successful completion.
WSIO_ERROR An error occurred.
WSIO_INVALID_PARM Invalid value for type, hndl, or vec_info, or invalid vector range specified by first and num parameters.
WSIO_NOT_SUPPORTED The platform does not have support for MSI/MSI-X features.

CONSTRAINS

None

EXAMPLES

wsio_ret_code_t ret;
wsio_msi_hndl_t msi_hndl;
spu_t vector_info[10];
wsio_msi_type_t object_type;

/* Retrieve the CPU identifiers of the first 10 vectors */
ret = wsio_msi_query(msi_hndl, WSIO_MSI_CPU_ID, 0, 10, (void *)vector_info);

/* Retrieve the type of the interrupt object */
ret = wsio_msi_query(msi_hndl, WSIO_MSI_OBJ_TYPE, 0, 1, (void *)&object_type);

SEE ALSO

wsio_msi_alloc(9F), wsio_msi_assign(9F), wsio_msi_capability(9F), wsio_msi_disable(9F),
wsio_msi_enable(9F), wsio_msi_free(9F), wsio_msi_get_cpus(9F), wsio_msi_resize(9F)
NAME

wsio_msi_resize - Resize the vector table of an MSI-X interrupt object.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_msi_resize (  
    wsio_msi_hndl_t *hndl,  
    uint64_t *new_size  
);

PARAMETERS

hndl Handle of an MSI-X interrupt object.

new_size Requested new size of the vector table. This will be updated to the actual allocated new size.

DESCRIPTION

The wsio_msi_resize service is called to change the number of vectors associated with an MSI-X type interrupt object. The parameter hndl is the handle of the interrupt object and new_size indicates the desired new size of the vector table. If the call is successful the actual number of vectors in the new object will be returned in new_size.

If the number of vectors is being increased, wsio_msi_resize will attempt to allocate at least as many vectors as requested and may round up the request to a larger size. If the requested number of vectors exceeds the number supported for the PCI device (as reported by wsio_msi_capability), wsio_msi_resize will return an error.

If the number of vectors is being decreased, wsio_msi_resize must not be called in an interrupt context and the vectors being removed must be disabled; otherwise, an error will be returned. An error will also be returned if the resize operation would remove all vectors from the object. MSI type objects cannot be resized.

RETURN VALUES

WSIO_OK Successful completion.

WSIO_ERROR An error occurred.

WSIO_INVALID_PARM The hndl is invalid, or new_size is NULL.

WSIO_NOT_SUPPORTED The platform does not have support for MSI/MSI-X features.

CONSTRAINTS

Do not call this service while holding a spinlock.

Do not called in an interrupt context, if the number of vectors is being decreased.

EXAMPLE

wsio_ret_code_t ret;
int vector_cnt, max_cnt;
wsio_msi_hndl_t msi_hndl;

/* Query the maximum number of MSI-X vectors for this device */
ret = wsio_msi_capability(my_isc, WSIO_MSI_X_CAPABILITY, &max_cnt);

/* Retrieve the current number of vectors in the object */
ret = wsio_msi_query(msi_hndl, WSIO_MSI_VECTOR_CNT, 0, 0, &vector_cnt);
/ * Increase the number of vectors by 5 */
vector_cnt += 5;

if (vector_cnt <= max_cnt) {
    /* Resize the object */
    ret = wsio_msi_resize(hndl, vector_cnt);
    if (ret == WSIO_OK) {
        printf("resized vector table to %d\n", vector_cnt);
    }
}

SEE ALSO
wsio_msi_alloc(9F), wsio_msi_assign(9F), wsio_msi_capability(9F), wsio_msi_disable(9F),
wsio_msi_enable(9F), wsio_msi_free(9F), wsio_msi_get_cpus(9F), wsio_msi_query(9F)
WSIO_ORDERED_INTERRUPTS(9F)

NAME

WSIO_ORDERED_INTERRUPTS--Macro to indicate whether interrupts are ordered with respect to DMA transactions.

SYNOPSIS

#include <sys/wsio.h>

int WSIO_ORDERED_INTERRUPTS(
    struct isc_table_type *isc
);

PARAMETERS

isc Pointer to the driver's isc_table entry.

DESCRIPTION

The WSIO_ORDERED_INTERRUPTS macro reports to a device driver about whether interrupts are ordered with respect to DMA transactions. If true (1) is returned, nothing needs to be done. However, if false (0) is returned, interrupts are not ordered, and drivers must ensure DMA transactions have completed by reading a status register, or by calling wsio_io_sync.

The WSIO_ORDERED_INTERRUPTS macro can be called in a non-blocking context.

RETURN VALUES

Returns a one (1) if interrupts are ordered with respect to DMA transactions, and a zero (0) otherwise.

CONSTRAINTS

None

EXAMPLE

if (WSIO_ORDERED_INTERRUPTS(isc_entry)) {
    wsio_io_sync(isc_entry);
} else {
    /* No syncing necessary */
}

SEE ALSO

wsio_io_sync(9F)
wsio_node_get_isc(9F)

NAME

wsio_node_get_isc - Retrieve the ISC structure of an I/O node.

SYNOPSIS

#include <sys/wsio.h>

struct isc_table_type *wsio_node_get_isc(
    void *handle
);

PARAMETERS

handle Pointer to the handle of the I/O node.

DESCRIPTION

The wsio_node_get_isc WSIO service is called to retrieve the WSIO isc structure associated with an I/O node. The I/O node must a T_INTERFACE or T_VIRTBUS type and must be claimed by a WSIO driver. Otherwise, the service returns NULL. This service can be combined with class_get_node to get the isc structure of an interface card given the class instance number.

RETURN VALUES

A pointer to an isc structure if the handle is a valid WSIO interface node. Otherwise, NULL value.

CONSTRAINTS

None

EXAMPLE

node = class_get_node(class, instance);

my_isc = wsio_node_get_isc(node);

SEE ALSO

wsio_async_scan(9F), wsio_claim_node(9F), wsio_destroy_legacy(9F), wsio_get_drv_priv(9F), wsio_put_drv_priv(9F), wsio_reg_legacy(9F), wsio_reg_node(9F), wsio_reg_node_t(9S)
NAME
wsio_port_inXX - Read xx bits from an I/O port.

SYNOPSIS
#include <sys/wsio.h>

void wsio_port_inXX(
    struct isc_table_type *isc,
    wsio_addr_handle_t handle,
    uint32_t addr,
    uintXX_t *data
);

PARAMETERS
isc Pointer to the driver's isc_table entry.
handle Handle obtained from wsio_map_port.
addr Offset into the port to read from.
data Contains the data read upon completion.

DESCRIPTION
The wsio_port_inXX WSIO function is used to read xx from an I/O port. The port must have
been mapped prior to the read by using the wsio_map_port macro. No endian swapping is
performed for port access, so if the local bus and the host bus are of opposite endianness, the
driver must perform an endian swap. The xx refers to the size of the transfer to perform and
must be either 8, 16, 32, or 64.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
wsio_addr_handle_t port_handle;
wsio_iop_t ioports_array[10];
uint32_t data;

/* An array with enough space for all ports needs to be
 * allocated. We have just made it a local variable in this
 * example */

if (wsio_get_ioports(isc_entry,10,ioports_array) != WSIO_OK) {
    /* There was a problem obtaining the ports */
    return(ERROR);
}

if (wsio_map_port(isc_entry,ioports_array[0].addr,
    ioports_array[0].size, &port_handle
) != WSIO_OK) {
    /* There was an error mapping the port */
    return(ERROR);
}

/* Now read 32-bits from the port */
wsio_port_in32(isc_entry, port_handle, 0, &data);

/* 'data' will now contain whatever was at offset 0 of I/O port 0 */

SEE ALSO

wsio_get_ioports(9F), wsio_map_port(9F), wsio_port_outXX(9F), wsio_unmap_port(9F)
NAME

wsio_port_outXX - Write XX bits to an I/O port.

SYNOPSIS

#include <sys/wsio.h>

void wsio_port_outXX(
    struct isc_table_type *isc,
    wsio_addr_handle_t handle,
    uint32_t addr,
    uintXX_t data
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
handle Handle obtained from wsio_map_port.
addr Offset into the port to read from.
data Data to be written.

DESCRIPTION

The wsio_port_outXX WSIO function is used to write XX bits to an I/O port. The port must have been mapped prior to the write by using the wsio_map_port macro. No endian swapping is performed for port access, so if the local bus and the host bus are of opposite endianness, the driver must perform an endian swap. The XX refers to the size of the transfer to perform, and must be either 8, 16, 32, or 64.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

wsio_addr_handle_t port_handle;
wsio_iop_t iports_array[10];
uint32_t data;

/* An array with enough space for all ports needs to be
 * allocated. We have just made it a local variable in this
 * example */

if (wsio_get_iports(isc_entry,10,iports_array) != WSIO_OK) {
    /* There was a problem obtaining the ports */
    return(ERROR);
}

if (wsio_map_port(isc_entry,iports_array[0].addr,
    iports_array[0].size,
    &port_handle) != WSIO_OK) {
    /* There was an error mapping the port */
    return(ERROR);
}

/* Now write 32-bits to the port */
wsio_port_out32(isc_entry, port_handle, 0, 0x5a);

/* This example wrote the 32 bits 0x5a to I/O port 0 at
 * offset 0 */

**SEE ALSO**

wsio_get_ioports(9F), wsio_map_port(9F), wsio_port_outXX(9F), wsio_unmap_port(9F)
NAME
wsio_probe_dev_info - WSIO device probe information.

SYNOPSIS
#include <sys/wsio.h>

struct wsio_probe_dev_info {
    unsigned short instance;
    unsigned short target;
    unsigned short opt_1;
    unsigned short opt_2;
    unsigned short opt_3;
};

DESCRIPTION
The wsio_probe_dev_info structure contains some of the device file information needed for
WSIO driver probe routines. This structure communicates hardware path information within
driver probe routines (especially for those cases where drivers may split their probe routines
into two routines; one for determining the next address to probe and one to actually try to build
and open a device file for that address). The hardware addressing information in this format can
easily be passed between two routines and is needed for building special device files for the
devices being probed.

STRUCTURE MEMBERS
instance Instance number of the nearest interface card ancestor.
target Relative hardware address of first layer to be probed.
opt_1 Optional. Sometimes used for the hardware address of the second layer to be
          probed (for example, LUN).
opt_2 Driver-discretionary element.
opt_3 Driver-discretionary element.

RETURN VALUES
None

EXAMPLE
A SCSI probe example might consist of two routines. The scsi_probe_function function
determines the next address to be probed and scsi_probe builds device files and actually tries
to open devices. Addressing information about the current node we’re trying to probe is passed
between the two routines via this structure. Elements of the structure are used by scsi_probe
to build device files as follows:

dev = ((major_num << 24) & 0xff000000);
dev |= ((probe_dev->instance << 16) & 0x00ff0000);
dev |= ((probe_dev->target << 12) & 0x0000f000);
dev |= ((probe_dev->opt_1 << 8) & 0x00000f00);

if ( (sctl_open(dev)) == 0 ) {
    /* Do an ioctl() on the device to get the ID information
    * for building the name, description, and id strings.
*/
}

SEE ALSO

wslio_register_probe_func(9F)
NAME

wsio_put_drv_priv - Save private driver information with an I/O node.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_put_drv_priv(
    void *handle,
    void *priv_info
);

PARAMETERS

handle Pointer to the handle of the I/O node with which to save the private data.
priv_info Pointer to the private data.

DESCRIPTION

The wsio_put_drv_priv WSIO service is called to save a reference to driver private data with an I/O node. This service does not copy the data; it saves a reference to it. It can be called in a non-blocking context.

RETURN VALUES

WSIO_OK The private information was successfully saved.
WSIO_INVALID_HNDL The caller specified an invalid I/O handle parameter.

CONSTRAINTS

None

EXAMPLE

wsio_legacy_info_t leg_node;

bzero((caddr_t)&leg_node, sizeof(wsio_legacy_info_t));
wsio_get_legacy(new_node, &leg_node);

while (leg_node->node != NULL)
{
    printf("name [%s]: desc[%s]\n", leg_node->name, leg_node->desc);
    wsio_get_legacy(new_node, &leg_node);
}

SEE ALSO

wsio_async_scan(9F), wsio_claim_node(9F), wsio_destroy_legacy(9F), wsio_get_drv_priv(9F),
wsio_node_get_isc(9F), wsio_reg_legacy(9F), wsio_reg_node(9F), wsio_reg_node_t(9S)
NAME

wsio_query_supported_function - Return a pointer to a function supported by WSIO.

SYNOPSIS

#include <sys/wsio.h>

void *wsio_query_supported_function(
   wsio_func_ptr_type_t func_id
);

PARAMETERS

func_id      Identifies what function pointer a driver is interested in.

<table>
<thead>
<tr>
<th>func_id</th>
<th>Function pointers returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_GET_HANDLER_REG_FUNC</td>
<td>WSIO function to register a driver's event handler.</td>
</tr>
<tr>
<td>WSIO_GET_HANDLER_UNREG_FUNC</td>
<td>WSIO function to unregister a driver's event handler.</td>
</tr>
<tr>
<td>WSIO_GET_INSTALL_DRV_FUNC</td>
<td>WSIO function to register a driver's function.</td>
</tr>
<tr>
<td>WSIO_GET_MASK_REG_FUNC</td>
<td>WSIO function to register a driver's supported event mask.</td>
</tr>
</tbody>
</table>

DESCRIPTION

The wsio_query_supported_function WSIO function queries WSIO to see if certain functions are supported. The purpose is that a driver can be installed on systems with different versions of wsio (11i or newer.) By doing this query, a driver can determine what functionality is supported and tailor its capabilities. If a function is supported, a pointer will be returned (otherwise, a NULL); a driver can then call this function pointer. This mechanism allows a driver to be able to link on multiple versions of wsio (11i or newer.)

RETURN VALUES

Returns a function pointer if supported; otherwise, a NULL.

CONSTRAINTS

None

EXAMPLE

static wsio_drv_info_t my_drv_info {
   ...
};

void my_event_handler()
{
   ...
}

void my_install(void)
{
   wsio_install_handler_func_ptr_t handler_reg_ptr;

   if (wsio_install_driver(&my_drv_info) {
      /* Query wsio to get a pointer to its registration *
         * function */
      handler_reg_ptr = wsio_query_supported_function(
         WSIO_GET_HANDLER_REG_FUNC);
      if (handler_reg_ptr) {
         ...
      }
   }
}
/* Now register our event handler with wsio */
ret = handler_reg_ptr(&my_drv_info,
    my_event_handler);
    ...
} ...
} ...

SEE ALSO
wsio_install_drv_event_handler(9F), wsio_install_drv_func(9F), wsio_reg_drv_capability_mask(9F),
wsio_uninstall_drv_event_handler(9F)
wsio_read_regXX(9F)

NAME

wsio_read_regXX - Read XX bits from a mapped device register.

SYNOPSIS

#include <sys/wsio.h>

void wsio_read_regXX(
    struct isc_table_type *isc,
    wsio_reg_info_t *reg_info,
    uint32_t offset,
    uintXX_t *data
);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
reg_info Pointer to information about the register to be read from. This register must have been successfully mapped by a call to wsio_map_reg.
offset The offset into the register to read from.
data Pointer to an appropriately sized data structure. This will contain the data upon completion of the read. The size must be selected based upon what size read is desired.

DESCRIPTION

The wsio_read_regXX WSIO functions are called by device drivers to read XX bits from a device register. The reg_info variable specifies which register to read from and offset indicates the correct location to read from. The value XX refers to 8, 16, 32, or 64, and indicates the amount of data to read. No endian swapping is performed for register reads. Therefore, if the local bus and the host bus are of opposite endianness, the driver must perform an endian swap.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

wsio_reg_info_t *registers;
uint32_t data;

registers = wsio_get_all_registers(isc_entry);
if (registers == NULL) {
    /* No registers exist. Return an error */
    return(ERROR);
}

/* All of the devices registers are now contained in * the 'registers' variable. They may be mapped as follows: */
if (wsio_map_reg(isc_entry,&registers[1]) != WSIO_OK) {
    return(ERROR);
}

/* The second device register (index 1 into the array) will * now be mapped. */
wsio_read_reg32(isc_entry, &registers[1],
(uint32_t)0x10, &data);
/* 'data' will now contain 32 bytes obtained from offset
 * 0x10 into the second register.
 */

SEE ALSO
wsio_get_all_registers(9F), wsio_map_reg(9F), wsio_unmap_reg(9F), wsio_write_regXX(9F)
NAME
wsio_reg_drv_capability_mask - Register a driver's capability mask.

SYNOPSIS
#include <sys/wsio.h>

int wsio_reg_drv_capability_mask(
    struct isc_table_type *isc,
    wsio_event_mask_t event_mask
);

PARAMETERS
isc Pointer to the driver's isc_table entry.
event_mask A mask representing the operations the driver will deal with in its event handler.

DESCRIPTION
The wsio_reg_drv_capability_mask WSIO function is called by a device driver to register with WSIO what operations the driver is capable of handling. The mask is an OR of wsio_events_t and is for an instance of the driver. This function must be called in a driver's attach routine after isc_claim is executed. This function can be called more than once to disable and enable certain functionalities. A new mask will supersede a previous one. If a platform does not support event handling, WSIO_HA_NA will be returned. In this case, a driver must ignore the error.

RETURN VALUES
WSIO_OK Returned on success.
WSIO_INVALID_COMBIN_EVENTS Invalid combination of events.
WSIO_HA_NA Platform does not support HA events.
WSIO_INVALID_EVENT Invalid event.
WSIO_INVALID_ISC Invalid isc.
WSIO_NO_DRV_HANDLER A driver's event handler has not been installed.

CONSTRAINTS
None

EXAMPLE
my_driver_attach() {
    int ret;
    wsio_event_mask_t my_mask = WSIO_EVENT_SUSPEND |
        WSIO_EVENT_RESUME;
    isc_claim(isc, &my_drv_info);
    ret = wsio_reg_drv_capability_mask(isc, my_mask);
    if ( (ret != WSIO_OK) && (ret != WSIO_HA_NA) ) {
        isc_unclaim(isc, &my_drv_info);
        free resource as needed
    }
    run the rest of the attach chain
}

SEE ALSO
wsio_install_drv_event_handler(9F), wsio_query_supported_function(9E)
NAME

wsio_register_addr_probe - Register a driver probe function.

SYNOPSIS

#include <sys/wsio.h>

void wsio_register_addr_probe(
  int (*func)(),
  char *drv_name
);

PARAMETERS

func A pointer to the driver probe function.
drv_name An ASCII string indicating the name of the driver.

DESCRIPTION

The wsio_register_addr_probe WSIO service is used to register an interface driver's probe function. The probe function is used by WSIO SCAN to look for I/O devices underneath interface cards claimed by the driver. The drv_name parameter must match the name field of the driver's drv_info_t structure.

The driver probe function must have the following calling syntax:

drv_addr_probe( void *handle,
  int (*dev_probe)(),
  drv_info_t *drv_info,
  void *probe_id,
  hw_path_t *hw_path,
  struct isc_table_type *isc,
  int probe_type,
  char *name,
  char *desc )

handle A pointer to a GIO structure. Drivers must not touch this structure.
dev_probe A pointer to a probe function registered via the WSIO service wsio_register_dev_probe if one exists, else NULL.
drv_info A pointer to the driver's drv_info_t structure.
probe_id A unique identifier for the device found.
hw_path When an input, the hardware path of the last device found. When an output, the hardware path of the next device to be found.
isc A pointer to the isc_table_type structure of the interface card being probed.
probe_type One of three types of probe, which are:
  PROBE_FIRST Find the first device underneath the interface card.
  PROBE_NEXT Find the next device after the previous one found.
  PROBE_ADDRESS Look for a device at the hardware address specified in hw_path.
name A pointer to a string initialized with the device's name such as scsi_disk. This information is used to match the device to a driver based on the information in the drv_path field of the wsio_drv_data_t structure.
desc A pointer to a string with a description of the device.

When the driver probe function is called with a probe_type of PROBE_FIRST the function must find the first device underneath the interface card specified by the isc parameter. The hw_path
parameter has the address of the interface card. When the driver probe function is called with a
probe_type of PROBE_NEXT, the driver must find the next device after the last device found. The
address of the last device is passed in the hw_path parameter. The driver then updates the hw_path
parameter with the address of the new device. Each time the probe function reports a device it
must return the additional information of probe_id, name and desc. The probe_id is a unique
identifier. The name string must match the drv_path field in the wsio_drv_data_t structure
of the driver that controls the device. desc is an ASCII string describing the device.

A probe function registered via wsio_register_addr_probe can be used as a standalone
probe function or in conjunction with another probe function registered by the service
wsio_register_dev_probe. In the latter case, the probe function registered via
wsio_registered_dev_probe is passed in as the second parameter to the first.

This method can be used by a driver stack where a device driver knows the syntax to talk to
certain devices and an interface driver would know the range of addresses for a given I/O bus.
The interactive driver would register an address probe function via
wsio_register_addr_probe and the device driver would register its probe function via
wsio_register_dev_probe

If the probe function is used as a standalone probe function then a NULL value is passed in as
the second parameter. Most drivers need only register a single probe function using the WSIO
service wsio_register_dev_probe.

RETURN VALUES
None

EXAMPLE

void mydrv_install()
{
    (void)wsio_register_addr_probe(mydrv_probe, "mydrv");
    wsio_install_driver(&mydrv_wsio_info);
}

CONSTRAINTS

SEE ALSO

drv_info(9S), wsio_drv_data_t(9S), wsio_register_dev_probe(9F)
NAME

wsio_register_dev_probe - Register a driver probe function.

SYNOPSIS

#include <sys/wsio.h>

int wsio_register_dev_probe ( int type,
                                int (*func) (),
                                char *drv_name
                           );

PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Indicates the driver data that the third parameter must match. Valid values are:</td>
</tr>
<tr>
<td>IF_CLASS</td>
<td>The third argument drv_name is to be matched with the drv_path field of the wsio_drv_data_t structure.</td>
</tr>
<tr>
<td>DRV_NAME</td>
<td>The third argument, drv_name is to be matched with the name field of the drv_info_t structure.</td>
</tr>
<tr>
<td>func</td>
<td>A pointer to the driver probe function.</td>
</tr>
<tr>
<td>drv_name</td>
<td>An ASCII string indicating the name or class of the driver.</td>
</tr>
</tbody>
</table>

DESCRIPTION

The wsio_register_dev_probe WSIO service is used to register a driver probe function. The driver probe function is used by WSIO SCAN to look for I/O devices beneath specific interface cards. Which cards to scan depend on the values of the first and third parameters. The third parameter, drv_name, is an ASCII string that is used to match the probe function to specific driver/interfaces cards. The first parameter, type, is used to indicate what driver information the ASCII string is to be matched to. If the parameter has the value IF_CLASS, it indicates the string must be matched to the drv_path field of the driver's wsio_drv_data_t structure. If the type parameter is set to the value of DRV_NAME, the third argument is matched with the name field of the driver's drv_info_t structure. A value of DRV_NAME causes a tight pairing of the probe function to a particular driver since the probe is matched to the driver's name. A value of IF_CLASS is more general since several drivers may have the same drv_path. Probe functions registered via the wsio_register_dev_probe service must have the following calling syntax:

```
drv_probe( void *handle,
            drv_info_t *drv_info,
            void *probe_id,
            hw_path_t *hw_path,
            struct isc_table_type * isc,
            int probe_type,
            char *name,
            char *desc )
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>A pointer to an internal GIO structure. Drivers must not attempt to access it.</td>
</tr>
<tr>
<td>drv_info</td>
<td>A pointer to the drv_info_t structure.</td>
</tr>
<tr>
<td>probe_id</td>
<td>A unique identifier for the device found.</td>
</tr>
<tr>
<td>hw_path</td>
<td>When an input, the hardware path of the last device found. When an output, the hardware path of the next device to be found.</td>
</tr>
<tr>
<td>isc</td>
<td>A pointer to the isc_table_type structure of the interface card being probed.</td>
</tr>
<tr>
<td>probe_type</td>
<td>The type of probe. The following types are supported:</td>
</tr>
<tr>
<td></td>
<td>PROBE_FIRST Find the first device underneath the interface card.</td>
</tr>
</tbody>
</table>
PROBE_NEXT  Find the next device after the previous one found as indicated by the hw_path parameter.

PROBE_ADDRESS  Look for a device at the specific hardware address.

name  A pointer to a string initialized with the device’s name such as scsi_disk. This information is used to match the device to a driver on the information in the drv_path.

desc  A pointer to a string with the device description. This is driver dependent.

When the driver probe function is called with a probe_type of PROBE_FIRST, the function must find the first device underneath the interface card specified by the isc parameter. The hw_path parameter has the address of the interface card. When the driver probe function is called with a probe_type of PROBE_NEXT the driver must find the next device after the last device found. The address of the last device is the last element of the hw_path parameter. The driver then updates the hw_path with the address of the new device. Each time the probe function reports a device it must return the additional information of probe_id, name, and desc. The probe_id is a unique identifier. The name string must match the drv_path field in the wsio_drv_data_t structure of the driver that controls the device. The desc parameter is an ASCII string describing the device.

RETURN VALUES

0  Successful completion.
-1  Error

CONSTRAINTS

None

EXAMPLE

```c
void mydrv_install()
{
    (void)wsio_register_dev_probe(DRV_NAME, mydrv_probe, "mydrv");
    wsio_install_driver(&mydrv_wsio_info);
}
```

SEE ALSO

drv_info(9S), wsio_drv_data_t(9S), wsio_register_addr_probe(9F)
**NAME**

wsio_reg_legacy - Register or validate a legacy (SCSI-2) I/O node.

**SYNOPSIS**

```c
#include <sys/wsio.h>

wsio_ret_code_t wsio_reg_legacy(
    void *handle,
    wsio_legacy_info_t *node_info
);
```

**PARAMETERS**

- `handle` Pointer to the handle of the node with which to register the legacy node.
- `node_info` Pointer to the `wsio_legacy_info_t` structure that contains the node information, including the type of node, the `drv_info_t` structure of the claiming driver and callback function, argument, and event mask.

**DESCRIPTION**

The `wsio_reg_legacy` WSIO service is used to register a legacy I/O node or to validate the existence of one. WSIO SCAN uses this function to register or validate a legacy I/O node.

The caller passes in the handle of the SCSI-3 node and a pointer to a `wsio_legacy_info_t` structure with the information on the legacy node. The caller must initialize the following fields in the `wsio_legacy_info_t` structure:

- `version` Set to `WSIO_VERSION_0`.
- `hw_path` The legacy hardware path.
- `type` The legacy node type. Valid values are `T_INTERFACE`, `T_VIRTBUS`, and `T_DEVICE`.
- `drv_info` The `wsio_drv_info_t` structure of the legacy driver.
- `name` A pointer to a character string with the property name of the node.
- `desc` A pointer to a character string with the property description of the node.
- `id` The id bytes of the legacy IO node.

On return, the `wsio_legacy_info_t` structure contains the following information about the legacy node:

- `node` The handle of the legacy I/O node.
- `instance` The class instance number assigned to the legacy node. This must be less than 256.

The `wsio_reg_node` routine first checks to see if the node already exists. If it does not, it registers a new I/O node and inserts it into the I/O tree at the specified legacy hardware path. The new node is flagged as a legacy I/O node. The `wsio_reg_node` routine also creates any ancestor nodes as transparent nodes, if specified by the hardware path. Therefore, the caller must not create them explicitly. The legacy information provided in the structure referenced by `node_info` will also be associated with the node. The `name` field is an optional string that is used to create the name property on the IO node.

If an I/O node already exists, `wsio_reg_node` validates it to see if it is the same as indicated by the `drv_info` pointer (using the `drv_info_t` structure and type). If they are the same, it overwrites all information with the new information and returns WSIO_FOUND_SAME. If the nodes are different, `wsio_reg_node` returns WSIO_FOUND_DIFF.

If `drv_info` is set to NULL, the node is registered in the UNCLAIMED state.
RETURN VALUES

WSIO_NEW_NODE  A new node was registered.
WSIO_FOUND_DIFF  The node exists, but does not match the criteria passed in.
WSIO_FOUND_SAME  The node exists and matches the criteria passed in.
WSIO_INVALID_HNDL  The caller specified an invalid handle parameter.
WSIO_INVALID_PARAM  The caller specified invalid hw_path information.
WSIO_ERROR  The caller specified an instance number greater than 255, or an error occurred.

CONSTRAINTS

None

SEE ALSO

wsio_async_scan(9F), wsio_claim_node(9F), wsio_destroy_legacy(9F), wsio_get_drv_priv(9F),
wsio_get_legacy(9F), wsio_node_get_isc(9F), wsio_put_drv_priv(9F), wsio_reg_legacy(9F),
wsio_reg_node_t(9S)
NAME

wsio_reg_node - Register or validate an I/O node.

SYNOPSIS

#include <sys/wsio.h>

wsio_ret_code_t wsio_reg_node(
    void *parent,
    hw_path_t *hw_path,
    wsio_reg_node_t *node_info,
    void **node
);

PARAMETERS

parent Pointer to the handle of the parent I/O node.

hw_path A pointer to the hardware path of the new node relative to the parent node.

node_info Pointer to the wsio_reg_node_t structure that contains the node information,
            including the type of node, the drv_info_t structure of the claiming driver and
            callback function, argument, and event mask.

node On return, the handle of the I/O node.

DESCRIPTION

The wsio_reg_node WSIO service is used to register or validate an I/O node. WSIO SCAN
uses this function to register or report an I/O node. If the node does not exist, a new one is
registered. Otherwise, if the node already exists, the node is validated within the context of the
I/O scan.

If the caller wants to create a transparent I/O node, set the type to T_CDIO_PRIVATE. Otherwise,
the caller must provide the drv_info_t structure of the claiming driver and set type to one
of the following I/O node types:

T_INTERFACE An interface I/O node.
T_DEVICE A device I/O node.
T_CDIO_PRIVATE A transparent I/O node.
T_VIRTBUS A virtual bus I/O node.
T_TGT_PATH A target path I/O node.
T_LUN_PATH A lunpath I/O node.

In addition, the caller must also provide a callback function in the cb_func and event_mask
fields of the node_info structure indicating what types of events the caller is interested in.

The wsio_reg_node routine first checks to see if the node already exists. If it does not, it registers
a new I/O node with the given information.

If an I/O node already exists, wsio_reg_node validates it to see if it is the same as indicated by
the node_info structure (using the drv_info_t structure and type). If they are the same, it returns
the status WSIO_FOUND_SAME. If the nodes are different, wsio_reg_node updates the I/O
node with the new information and returns WSIO_FOUND_DIFF.

If the caller specifies the T_VIRTBUS type, it is registering an I/O node underneath the virtual
root. The service ignores the parent parameter and attempts to find an existing virtual bus I/O
node that matches the driver name and hardware path. If a match is found, it returns the node
handle; otherwise, it creates a new I/O node of T_VIRTBUS type and assigns the lowest available
hardware address to it.
I/O nodes are destroyed with a call to `io_destroy`, which in turn calls the I/O node callback.

**RETURN VALUES**

- **WSIO_NEW_NODE** A new node was registered.
- **WSIO_INVALID_PARM** The caller specified an invalid parameter.
- **WSIO_FOUND_DIFF** The node exists, but does not match the criteria passed in.
- **WSIO_FOUND_SAME** The node exists and matches the criteria passed in.
- **WSIO_ERROR** An error occurred.

**CONSTRAINTS**

None

**SEE ALSO**

`wsio_async_scan(9F)`, `wsio_claim_node(9F)`, `wsio_destroy_legacy(9F)`, `wsio_get_drv_priv(9F)`, `wsio_get_legacy(9F)`, `wsio_node_get_isc(9F)`, `wsio_put_drv_priv(9F)`, `wsio_reg_legacy(9F)`, `wsio_reg_node_t(9S)`
NAME

wsio_reg_node_t - WSIO node registration structure.

SYNOPSIS

#include <sys/wsio.h>

DESCRIPTION

The wsio_reg_node_t structure WSIO structure type, defined in <sys/wsio.h>, is used to register or validate an I/O node. It is passed as a parameter to wsio_reg_node or wsio_reg_legacy.

STRUCTURE MEMBERS

typedef struct wsio_reg_node {
    int version;
    wsio_drv_info_t *drv_info;
    int type;
    int (*cb_func())();
    void *cb_arg;
    io_events_t events_mask;
    char *name;
    char id[16];
    char *desc;
    struct isc_table_type *isc
} wsio_reg_node_t;

version       Data structure version number.
drv_info      A pointer to the drv_info structure.
type          Type of I/O node created.
cb_func       I/O node callback function.
cb_arg        Optional argument for the callback function.
events_mask   The events callback handles.
name          Name property for the node.
id            ID property for the node.
desc          Description property for the node.
isc           Pointer to an isc structure if type is T_INTERFACE.

SEE ALSO

wsio_async_scan(9F), wsio_claim_node(9F), wsio_destroy Legacy(9F), wsio_get_drv_priv(9F), wsio_get Legacy(9F), wsio_legacy_info_t(9S), wsio_node_get_isc(9F), wsio_put_drv_priv(9F), wsio_reg Legacy(9F), wsio_reg_node(9F)
wsio_remap_dma_buffer(9F)

NAME

wsio_remap_dma_buffer -- Map pre-allocated IOVAs to new host ranges.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_remap_dma_buffer(
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_range_type_t range_type,
    wsio_dma_map_t *host_range,
    wsio_dma_map_t *io_range
);

PARAMETERS

isc        Pointer to the driver's isc_table entry.
dma_handle DMA handle allocated using wsio_allocate_dma_handle.
range_type Indicates the type of host memory being mapped. It can be:
    KERNELSPACE Indicates host_range is a kernel virtual buffer.
    PHYSICAL Indicates host_range is a physical buffer.
    > 0 Indicates host_range is in user space, and this will be the space
        ID of the virtual address.
host_range Pointer to an address/length structure that contains information about the host
        space to map. If the mapping was only partially completed, this will contain
        information about the remaining space to be mapped when the call completes.
io_range  Pointer to an address/length structure that will contain information about the
        I/O space that was mapped.

DESCRIPTION

The wsio_remap_dma_buffer WSIO function is called by a device driver to map a new host
memory address to existing I/O Virtual Addresses (IOVA). The IOVAs must have been previously
allocated via a call to wsio_map_dma_buffer, wsio_remap_dma_buffer, or
wsio_fastmap_dma_buffer.

The io_range must use exactly the same number of mapping resources as the previous mapping.
This can be ensured by making sure the buffers are page-aligned and of equal sizes.

All mappings will remain in effect until wsio_unmap_dma_buffer or
wsio_remap_dma_buffer are called to remove or change them.

A callback function is not necessary for wsio_remap_dma_buffer because DMA resources
were allocated when the initial mapping took place.

The wsio_remap_dma_buffer function can be called in a non-blocking context.

RETURN VALUES

WSIO_MAP_OK        Returned if the entire buffer has been mapped.
WSIO_MAP_E_HIGH_ADDR Returned if the call failed because the device cannot reach
                        the destination address.
WSIO_MAP_E_PARAMETER_ERROR Returned if an invalid parameter has caused failure of the
                                call.
WSIO_MAP_E_UNKNOWN_ERROR Returned for hardware or other errors.
CONSTRAINTS

None

EXAMPLE

```c
void *dma_handle;
wsio_map_context_t dma_con;
wsio_dma_map_t host_range, new_host_range, io_range;

dma_handle = wsio_allocate_dma_handle(isc_entry);
wsio_init_map_context(&dma_con);

host_range.iov_base = host_virtual_address;
host_range.iov_len = dma_buffer_length;

if (wsio_map_dma_buffer(isc_entry,dma_handle,
           dma_con,WSIO_DMA_OUTBOUND KERNELSPACE,
           &host_range,&io_range)!= WSIO_MAP_OK) { 
    return(ERROR);
}

new_host_range.iov_base = new_host_virtual_address;
new_host_range.iov_len = dma_buffer_length;

if (wsio_remap_dma_buffer(isc_entry,dma_handle,
               KERNELSPACE,&new_host_range, &io_range
               ) != WSIO_MAP_OK) {
    return(ERROR);
}

/* The host virtual buffer represented in the above
   code by 'host_virtual_address' was mapped. After that,
   a second host virtual buffer represented by
   'new_host_virtual_address' was mapped using the
   same I/O virtual address as the initial mapping.
   This example does not take into account a
   WSIO_MAP_W_PARTIAL return code being returned by the
   call to wsio_map_dma_buffer(). */
```

SEE ALSO

wsio_allocate_dma_handle(9F), wsio_allocate_shared_mem(9F), wsio_dma_pass_thru(9F),
wsio_fastmap_dma_buffer(9F), wsio_free_dma_handle(9F), wsio_free_shared_mem(9F),
wsio_flush_shared_mem(9F), wsio_init_map_context(9F), wsio_iova_to_phys(9F),
wsio_map_dma_buffer(9F), wsio_set_device_attributes(9F), wsio_set_dma_attributes(9F),
wsio_unmap_dma_buffer(9F)
NAME

wsio_set_description - Set the I/O tree node description for this driver.

SYNOPSIS

#include <sys/wsio.h>

void wsio_set_description(
    struct isc_table_type *isc,
    char *description
);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
description String containing the description.

DESCRIPTION

The wsio_set_description WSIO function sets the I/O tree node description of a driver. This functionality is provided because at times, WSIO can not make appropriate decisions as to how to describe a driver at driver install time. This description is used by the ioscan command that is provided to system users, and therefore is essential to describing the system.

Many drivers may not need to call this function. To decide if it is necessary, a driver writer must make a decision based upon whether the ioscan output for that device driver is understandable.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

wsio_set_description (isc_entry,"My Driver Description");

SEE ALSO
wsio_set_dma_attributes(9F)

NAME

wsio_set_dma_attributes - Associate DMA hints with a DMA handle.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_set_dma_attributes(
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_dma_attribute_t attribute,
    wsio_dma_attr_param_t param
);

PARAMETERS

isc Pointer to the driver's isc_table entry.

dma_handle DMA handle allocated using wsio_allocate_dma_handle.

attribute Indicates which hint to set for the device associated with dma_handle. The possible attributes are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_DMA_ATTR_ADDR_WIDTH</td>
<td>Bits of addressing supported by the device. This is used to determine whether a device can DMA directly to memory buffers. Default: 32</td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_ALIGNMENT</td>
<td>Byte alignment of DMA buffer required for device. Default: H/W Dependent</td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_ATM</td>
<td>ATM hint; used by hardware in some implementations.</td>
</tr>
<tr>
<td>0</td>
<td>Not ATM</td>
</tr>
<tr>
<td>1</td>
<td>ATM48 (optimize for 48-byte transfers)</td>
</tr>
<tr>
<td>2</td>
<td>ATM192 (optimize for 192-byte transfers)</td>
</tr>
<tr>
<td>Default: 0</td>
<td></td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_CALLBACK</td>
<td>Specifies a function to call when resources become available. Default: NULL</td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_CALLBACK_ARG</td>
<td>Specifies an argument to the callback function. Default: 0</td>
</tr>
<tr>
<td>WSIO_DMA_ATTR_FLUSH_ON_USE</td>
<td>Specifies to flush the cacheline from any intermediate buffers as soon as it is</td>
</tr>
</tbody>
</table>
referenced. This inhibits any coalescing of data by bus bridges.
Default: 0

WSIO_DMA_ATTR_IGN_ALIGN Specifies the mapping service must not handle cacheline fragments in a special way.
Default: 0

WSIO_DMA_ATTR_INTERLEAVE IOVA allocation model
0 DMA streams are normally interleaved (mass storage)
1 DMA streams are normally not interleaved (networking)
2 DMA buffers are static and accessed randomly (lowfat)
Default: 0

WSIO_DMA_ATTR_PREFETCH Specifies how aggressively hardware must prefetch for outbound DMA.
0 No prefetch
1 Moderate prefetch
2 Aggressive prefetch
Default: 1

WSIO_DMA_ATTR_SAFE Specifies the use of the most conservative coherency model for inbound DMA. Inhibits semi-coherent transactions such as WRITE_PURGE unless it is guaranteed that no data in processor caches will be lost.
1 On
2 Off
Default: 0

WSIO_DMA_ATTR_TXN_SIZE Specifies the default transaction size used by the device. This is used by hardware to optimize conversion of transactions between buses.
Default: Hardware dependent

WSIO_DMA_ATTR_INBOUND DMA buffers will be used exclusively for inbound DMA.
Default: 0

WSIO_DMA_ATTR_OUTBOUND DMA buffers will be used exclusively for outbound DMA.
Default: 0

WSIO_DMA_ATTR_STABLE Indicates that the buffer will not be modified by another entry while mapped for DMA.
1 True for data buffers
param   Driver-defined parameter passed as the first parameter to *isr*. Typically, *isc* is passed as *arg1*.

DESCRIPTION

The `wsio_set_dma_attributes` WSIO function is used to associate DMA transaction hints and attributes with a specific DMA handle. These hints override any hints set via `wsio_dma_set_device_attributes`, and are overridden by some hints passed in as parameters to `wsio_map_dma_buffer`. `wsio_set_dma_attributes` can be called in a non-blocking context.

If `WSIO_DMA_ATTR_INTERLEAVE` is set to 1, a subsequent `wsio_allocate_shared_mem` or `wsio_map_dma_buffer` can only successfully request a buffer of a maximum of one page (4K) in length and this buffer cannot cross a page boundary. This is the limitation placed by the underlying platform. If a larger buffer is desirable, use the default value of 0. This larger buffer can be used for control structures rather than packet DMAs.

Do not call `wsio_set_dma_attributes` to set the `WSIO_DMA_ATTR_INTERLEAVE` to the default value of 0. The call will fail.

RETURN VALUES

`WSIO_MAP_OK` Successful completion.

`WSIO_MAP_E_PARAMETER_ERROR` Returned if an invalid parameter has caused failure of the call.

CONSTRAINTS

None

EXAMPLE

```c
if (wsio_set_dma_attributes(isc_entry,dma_handle,
   WSIO_DMA_ATTR_INTERLEAVE,1) != WSIO_MAP_OK) {
   /* There was a parameter error */
   return(ERROR);
} else {
   /* DMA streams are now not normally interleaved for * all DMA associated with dma_handle */
   return(0);
}
```

SEE ALSO

`wsio_allocate_dma_handle(9F)`, `wsio_allocate_shared_mem(9F)`, `wsio_dma_pass_thru(9F)`,
`wsio_dma_set_device_attributes(9F)`, `wsio_fastmap_dma_buffer(9F)`, `wsio_free_dma_handle(9F)`,
`wsio_free_shared_mem(9F)`, `wsio_flush_shared_mem(9F)`, `wsio_init_map_context(9F)`,
`wsio_iova_to_phys(9F)`, `wsio_map_dma_buffer(9F)`, `wsio_remap_dma_buffer(9F)`,
`wsio_unmap_dma_buffer(9F)`
**NAME**

wsio_set_dma_callback - Set the callback function and argument for DMA.

**SYNOPSIS**

```c
#include <sys/wsio.h>

wsio_map_status_t wsio_set_dma_callback(
    struct isc_table_type *isc,
    void *dma_handle,
    void *func,
    void *arg
);
```

**PARAMETERS**

- **isc** Pointer to the driver's isc_table entry.
- **dma_handle** DMA handle allocated using wsio_allocate_dma_handle.
- **func** Function pointer to be used as a callback.
- **arg** Argument to be passed to the callback when it is called.

**DESCRIPTION**

The `wsio_set_dma_callback` WSIO function is called by a device driver to setup the callback function for certain DMA transactions. If resources are not available when `wsio_map_dma_buffer`, `wsio_fastmap_dma_buffer`, or `wsio_allocate_shared_memory` are called, and a callback function is setup, WSIO_MAP_W_CALLBACK is returned to the caller, and the callback function will be called when resources become available. This eliminates the need to continuously loop to attempt to obtain DMA resources.

**RETURN VALUES**

- **WSIO_MAP_OK** Successful completion.
- **WSIO_MAP_E_PARAMETER_ERROR** Returned if an invalid parameter has caused failure of the call.

**CONSTRAINTS**

None

**EXAMPLE**

```c
if (wsio_set_dma_callback(isc_entry, dma_handle,
    callback_func, callback_arg)
    != WSIO_MAP_OK) {
    /* There was a parameter error */
    return(ERROR);
}
```

**SEE ALSO**

wsio_allocate_dma_handle(9F), wsio_allocate_shared_mem(9F), wsio_dma_pass_thru(9F), wsio_fastmap_dma_buffer(9F), wsio_free_dma_handle(9F), wsio_free_shared_mem(9F), wsio_flush_shared_mem(9F), wsio_init_map_context(9F), wsio_map_dma_buffer(9F), wsio_remap_dma_buffer(9F), wsio_set_device_attributes(9F), wsio_set_dma_attributes(9F), wsio_unmap_dma_buffer(9F)
NAME
wsio_uninstall_driver - Uninstall a driver's header structure from the WSIO CDIO.

SYNOPSIS
#include <sys/wsio.h>
int wsio_uninstall_driver(
    wsio_drv_info_t *wsio_drv_info
);

PARAMETERS
wsio_drv_info Pointer to the driver's wsio_info_t structure.

DESCRIPTION
The wsio_uninstall_driver WSIO function uninstalls a driver's header structure from the
WSIO CDIO. The wsio_uninstall_driver function is called by the driver prior to unloading.

RETURN VALUES
0 Successful completion.
<>0 Error

CONSTRAINTS
None

SEE ALSO
wsio_install_driver(9F)
NAME

wsio_uninstall_drv_event_handler - Uninstall a driver's event handler.

SYNOPSIS

#include <sys/wsio.h>

int wsio_uninstall_drv_event_handler(
    wsio_drv_t *drv_info,
    wsio_drv_event_handler_t drv_handler
);

PARAMETERS

drv_info Pointer to the driver's wsio_drv_info_t structure.
drv_handler Function pointer to the driver's event handler.

DESCRIPTION

The wsio_uninstall_drv_event_handler WSIO function is called by a driver to uninstall its event handler. The call is made if a driver needs to clean up after an init failure, online deletion, or DLKM.

RETURN VALUES

WSIO_OK Returned on success.
WSIO_DRV_NOT_FOUND Could not find the driver (driver has not registered yet).
WSIO_ERROR Returned on failures.
WSIO_INFO_NULL Drv_info is NULL.
WSIO_NO_DRV_HANDLER Invalid drive handler.

CONSTRAINTS

None

EXAMPLE

static wsio_drv_info_t my_drv_info {
    &my_info,
    &my_ops,
    &my_data,
    WSIO_DRV_CURRENT_VERSION,
}

my_cleanup() {
    int ret;
    ret = wsio_uninstall_drv_event_handler(
        &my_drv_info, my_handler);
    ...
    ret = wsio_uninstall_driver(&my_drv_info);
}

SEE ALSO

wsio_install_drv_event_handler(9F)
NAME

wsio_unmap_cfg_handle - Release a configuration space handle.

SYNOPSIS

#include <sys/wsio.h>

int wsio_unmap_cfg(
    struct isc_table_type *isc,
    wsio_addr_handle_t *cfg_handle
);

PARAMETERS

isc Pointer to the driver's isc_handle entry.

cfg_handle Pointer to the configuration handle.

DESCRIPTION

The wsio_map_cfg_handle WSIO function is called by device drivers to release a configuration space handle.

RETURN VALUES

WSIO_OK Indicates a handle was successfully returned in cfg_handle.

WSIO_ERROR Indicates there was a parameter error and the handle has not been released.

CONSTRAINTS

None

EXAMPLE

wsio_addr_handle_t handle;

if (wsio_map_cfg_handle(isc_entry,&handle) != WSIO_OK) {
    /* Error obtaining configuration space handle. Return
       * an error code */
    return(ERROR);
}

if (wsio_unmap_cfg(isc_entry,&handle) != WSIO_OK) {
    /* Error releasing configuration space handle. Return an
       * error code */
    return(ERROR);
}

/* The configuration space handle is no longer valid */

SEE ALSO

wsio_map_cfg_handle(9F), wsio_cfg_inXX(9F), wsio_cfg_outXX(9F)
NAME

wsio_unmap_dma_buffer - Remove a DMA packet mapping.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_unmap_dma_buffer(
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_dma_map_t *io_range
);

PARAMETERS

isc             Pointer to the driver's isc_table entry.
dma_handle      DMA handle allocated using wsio_allocate_dma_handle.
io_range       Pointer to an address/length structure that contains the information about the mapping to remove.

DESCRIPTION

The wsio_unmap_dma_buffer WSIO function is called by a device driver to remove a packet DMA memory mapping, and to free all resources associated with such a mapping. The io_range must have been obtained via a previous call to wsio_map_dma_buffer, wsio_remap_dma_buffer, or wsio_fastmap_dma_buffer.

The wsio_unmap_dma_buffer function can be called in a non-blocking context.

RETURN VALUES

WSIO_MAP_OK       Successful completion.
WSIO_MAP_E_PARAMETER_ERROR Returned if an invalid parameter has caused failure of the call. The buffer will not be unmapped.

CONSTRAINTS

None

EXAMPLE

void *dma_handle;
wsio_map_context_t dma_con;
wsio_dma_map_t host_range, new_host_range, io_range;

dma_handle = wsio_allocate_dma_handle(isc_entry);
wsio_init_map_context(&dma_con);

host_range.iov_base = host_virtual_address;
host_range.iov_len = dma_buffer_length;

if (wsio_map_dma_buffer(isc_entry,dma_handle,
            dma_con,WSIO_DMA_OUTBOUND,KERNELSPACE,&host_range,
            &io_range)!= WSIO_MAP_OK) {
    return(ERROR);
}

if (wsio_unmap_dma_buffer(isc_entry,dma_handle,&io_range
        ) != WSIO_MAP_OK) {
/* There must have been a parameter error. */
    return(ERROR);
The host virtual buffer represented in the above code by 'host_virtual_address' was mapped and then immediately unmapped. This example does not take into account a WSIO_MAP_W_PARTIAL return code being returned by the call to wsio_map_dma_buffer().

SEE ALSO

wsio_allocate_dma_handle(9F), wsio_allocate_shared_mem(9F), wsio_dma_pass_thru(9F),
wsio_fastmap_dma_buffer(9F), wsio_free_dma_handle(9F), wsio_free_shared_mem(9F),
wsio_flush_shared_mem(9F), wsio_init_map_context(9F), wsio_iova_to_phys(9F),
wsio_map_dma_buffer(9F), wsio_remap_dma_buffer(9F), wsio_set_device_attributes(9F),
wsio_set_dma_attributes(9F)
wsio_unmap_port(9F)

NAME

wsio_unmap_port - Unmap an I/O port.

SYNOPSIS

#include <sys/wsio.h>

int wsio_unmap_port(
    struct isc_table_type *isc,
    int32_t port_addr,
    size_t size,
    wsio_addr_handle_t port_handle
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
port_addr Address of the port obtained from wsio_get_ioports.
size Size of the port.
port_handle Handle obtained from wsio_map_port.

DESCRIPTION

The wsio_unmap_port WSIO function removes the mapping done by wsio_map_port. After this is called, the port must not be accessed by the driver anymore.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_ERROR Parameter error.

CONSTRAINTS

None

EXAMPLE

wsio_addr_handle_t port_handle;
wsio_iop_t ioports_array[10];
/* An array with enough space for all ports needs to be * allocated */

if (wsio_get_ioports(isc_entry,10,ioports_array) != WSIO_OK) {
    /* There was a problem obtaining the ports */
    return(ERROR);
}

if (wsio_map_port(isc_entry,ioports_array[0].addr,
    ioports_array[0].size, &port_handle) != WSIO_OK) {
    /* There was an error mapping the port */
    return(ERROR);
}

/* Now unmmap the port */
if (wsio_unmap_port(isc_entry,ioports_array[0].addr,
    ioports_array[0].size,port_handle) != WSIO_OK) {
    /* There was an error unmapping the port */
    return(ERROR);
}
SEE ALSO

wsio_get_ioports(9F), wsio_map_port(9F), wsio_port_inXX(9F), wsio_port_outXX(9F)
NAME

wsio_unmap_reg - Unmap a device register.

SYNOPSIS

#include <sys/wsio.h>

int wsio_unmap_reg(
    struct isc_table_type *isc,
    wsio_reg_info_t *reg_info
);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
reg_info Pointer to information about the register to be unmapped. This must be the same structure used to map the register.

DESCRIPTION

The wsio_unmap_reg WSIO function removes the mapping done by wsio_map_reg.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_ERROR Could not unmap the register.

CONSTRAINTS

None

EXAMPLE

wsio_reg_info_t *registers;

registers = wsio_get_all_registers(isc_entry);
if (registers == NULL) {
    /* No registers exist. Return an error */
    return(ERROR);
}

if (wsio_map_reg(isc_entry,&registers[1]) != WSIO_OK) {
    return(ERROR);
}

/* The second device register (index 1 into the array) will now * be mapped.
 */
if (wsio_unmap_reg(isc_entry,&registers[1]) != WSIO_OK) {
    return(ERROR);
}

/* The second device register will now be unmapped */

SEE ALSO

wsio_get_all_registers(9F), wsio_map_reg(9F), wsio_read_regXX(9F), wsio_write_regXX(9F)
NAME

wsio_unregister_dev_probe - Unregisters a driver probe function.

SYNOPSIS

#include <sys/wsio.h>

int wsio_unregister_dev_probe(
    int type,
    char *name
);

PARAMETERS

type Indicates the driver data that the second parameter must match. Valid values are:

    IF_CLASS The second argument, name, is to be matched with the \textit{drv_path} field of the \texttt{wsio_drv_data_t} structure.

    DRV_NAME The second argument, name, is to be matched with the \textit{name} field of the \texttt{drv_info_t} structure.

name An ASCII string indicating the name or class of the driver.

DESCRIPTION

The \texttt{wsio_unregister_dev_probe} WSIO service is used to unregister a driver probe function that was previously registered by a call to \texttt{wsio_register_dev_probe}. The \textit{type} and \textit{name} parameters passed to \texttt{wsio_unregister_dev_probe} must be the same as the first and third arguments passed to \texttt{wsio_register_dev_probe} when the driver registered the probe function.

The first parameter, \textit{type}, is used to indicate what driver information the ASCII string is to be matched to. If the parameter has the value \texttt{IF_CLASS}, it indicates the string must be matched to the \textit{drv_path} field of the driver's \texttt{wsio_drv_data_t} structure. If the \textit{type} parameter is set to the value \texttt{DRV_NAME}, the second argument is matched with the \textit{name} field of the driver's \texttt{drv_info_t} structure. The second parameter, \textit{name}, is an ASCII string with the driver's name or path.

The service is used primarily by DLKM type drivers in their unload routines.

RETURN VALUES

0 Successfully found and deleted the driver.

-1 Not found.

CONSTRAINTS

None

EXAMPLE

int mydrvUnload( void *arg)
{
    int ret;
    struct isc_table_type *isc;
    void (token, *priv_ptr;

    /********************************************************************************
    * Remove the attach function from the DLKM attach list
    ********************************************************************************/
    if (mod_wsio_attach_list_remove (MOD_WSIO_CORE,
&module_name_core_attach))
    return (ENXIO);

/********************************************
* Unregister the device probe
*********************************************/
:void) wsio_unregister_dev_probe(IF_CLASS,"mydrv_path");
    "probe_name";

/********************************************
* Uninstall the driver. If it fails, go back to the
* load state and undo what has been done in the
* unload routine.
*********************************************/
if(wsio_uninstall_driver(&module_name_wsio_info)) {
    return (ENXIO);
}
    return(0);

SEE ALSO

wsio_drv_data_t(9S), wsio_drv_info(9S), wsio_register_dev_probe(9F)
NAME

wsio_write_regXX - Write XX bits to a mapped register.

SYNOPSIS

#include <sys/wsio.h>

void wsio_write_regXX(
    struct isc_table_type *isc,
    wsio_reg_info_t *reg_info,
    uint32_t offset,
    uintXX_t data
);

PARAMETERS

isc Pointer to the driver's isc_table entry.
reg_info Pointer to information about the register to be written to. This register must have
been successfully mapped by a call to wsio_map_reg.
offset The offset into the register to write to.
data Appropriately sized piece of data to be written. The size must be selected based
upon what size write is desired.

DESCRIPTION

The wsio_write_regXX WSIO functions are called by device drivers to write XX bits to a device
register. The reg_info variable specifies which register to write to, and offset indicates the correct
location to write to. The value XX refers to 8, 16, 32, or 64, and indicates the amount of data
to write. No endian swapping is performed for register writes. Therefore, if the local bus and the
host bus are of opposite endianness, the driver must perform an endian swap.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

wsio_reg_info_t *registers;
uint32_t data = 0x5a;

registers = wsio_get_all_registers(isc_entry);
if (registers == NULL) {
    /* No registers exist. Return an error */
    return(ERROR);
}
/* All of the devices registers are now contained in the
 * 'registers' variable. They may be mapped as follows:
 */
if (wsio_map_reg(isc_entry,&registers[1]) != WSIO_OK) {
    return(ERROR);
}
/* The second device register (index 1 into the array) will now
 * be mapped. */
wsio_write_reg32(isc_entry, &registers[1],
    (uint32_t)0x10, data);
/* The 32 bytes consisting of 0x5a will now be written to * offset 0x10 into the second register. /*

**SEE ALSO**

wsio_get_all_registers(9F), wsio_map_reg(9F), wsio_read_regXX(9F), wsio_unmap_reg(9F)
This chapter describes driver functions that are specific to PCI Services.

**PCI Macros and Functions**

In the function synopses, each parameter type is prefixed with the comment /*IN*/ or /*OUT*/, indicating that the parameter is input to or output from the function. These comments are included here because they are helpful and because they are used as keywords on other platforms.

**PCI Masters and Coherency**

Certain combinations of WSIO mapping service calls can interact with PCI masters to create an inconsistent view of memory. This behavior is documented in `pci_errata(9F)`. 
CONNECT_INIT_ROUTINE(9F)

NAME
CONNECT_INIT_ROUTINE - Associate an initialization routine with a driver.

SYNOPSIS
#include <sys/pci.h>

#define CONNECT_INIT_ROUTINE(isc, init_routine) \   
((isc)->gfsw->init = (init_routine))

PARAMETERS
isc Pointer to an ISC table associated with the device.
init Pointer to your driver's initialization routine.

DESCRIPTION
CONNECT_INIT_ROUTINE is a macro that associates a driver_if_init routine with a driver. It is typically used in a device's driver_attach routine. After all attach routines are executed, the driver_if_init routine is called by the kernel to perform further card and driver initialization.
If the driver performs all needed initialization in its driver_attach routine, a driver_if_init routine is not needed.
PCI_ATTACH_DEV_INIT_ERROR(9F)

NAME
PCI_ATTACH_DEV_INIT_ERROR - Report initialization error to WSIO Services.

SYNOPSIS
#include <sys/pci.h>

#define PCI_ATTACH_DEV_INIT_ERROR(isc) \ 
    (((struct wsio_if_info *)(isc)->if_info))->flags |= INIT_ERROR

PARAMETERS
isc Pointer to the ISC structure passed into your attach routine.

DESCRIPTION
PCI_ATTACH_DEV_INIT_ERROR is a macro that reports to WSIO Services that an error occurred during the device's initialization. It causes an error flag to be set in a structure in isc. It must be used as appropriate in the device's driver_attach or driver_if_init routine.
NAME

pci_desc_bus_transactions_isc - Describe the typical bus performance path transaction size.

SYNOPSIS

#include <sys/pci.h>

int pci_desc_bus_transactions_isc(
    struct isc_table_type *isc,
    BUS_TRANS_DESC *desc
);

PARAMETERS

isc Pointer to an ISC table associated with the device.
desc A pointer to a BUS_TRANS_DESC structure, defined as:

typedef struct  bus_trans_desc
{   
    uint32_t   read_width;
    uint32_t   write_width;
    uint8_t    reserved[20];
}  BUS_TRANS_DESC;

where:

read_width The number of 32-bit words in the width of the read path. A 0 value means do not change the current value.
write_width The number of 32-bit words in the width of the write path. A 0 value means do not change the current value.
reserved Reserved for future extensions.

DESCRIPTION

The pci_desc_bus_transactions_isc PCI function allows a driver to describe the bus transaction size of a card’s typical performance DMA accesses. Its use is entirely optional, since PCI Services provides a reasonable, general-purpose default.

The key to understanding how to use this routine for performance tuning is to recognize that the PCI bus supports variable-length data transactions. These transaction lengths may not map directly to transaction lengths on other buses on the system. Also, performance depends on other buses initiating the appropriate transaction in advance.

The purpose of the routine is to provide a hint of the typical performance path transaction size used by a specific card. While PCI can technically support unlimited transfer sizes (specifically, a dynamic number of data phases per PCI transaction), most PCI device/functions have some preferred size or can be programmed to use a particular size. By providing this hint, the PCI Services can, for some bus adapters, set up the bus adapter hardware to better map cycles between buses.

If you do not use pci_desc_bus_transactions_isc, PCI Services provide defaults that are intended to be safe and to give reasonable performance.

RETURN VALUES

PCI_OKAY The hints were used.
PCI_BUS_HINTS_BAD_DATA The BUS_TRANS_DESC structure is incorrect.
PCI_BUS_HINTS_NOT_USED The hints are not supported in this configuration.
CONSTRAINTS

None
NAME

cpci_get_port_hndl_isc(9F) - Obtain a system defined handle for manipulating a range of PCI I/O ports.

SYNOPSIS

#include <sys/pci.h>

int pci_get_port_hndl_isc(
    struct isc_table_type *isc,
    uint32_t *pci_io_addr,
    uint32_t size,
    PCI_PORT_HNDL *phndl
);

PARAMETERS

isc Pointer to an ISC table associated with the device.
pci_io_addr The address of a range of PCI I/O ports.
size The size of the PCI I/O ports.
phndl A pointer to the location where the routine is to place the handle provided by the system.

DESCRIPTION

The pci_get_port_hndl_isc PCI function obtains a system-defined handle for manipulating a range of PCI I/O ports.
The routine can block or sleep. Therefore, call it only in a thread context.

RETURN VALUES

0 Failure — a handle could not be returned by the system.
1 Success — the value pointed to by phndl is a valid handle.

CONSTRAINTS

None

EXAMPLES

The pci_read_cfg_uint32_isc routine reads the card's configuration space and retrieves the PCI address associated with an I/O port range. This address and size is passed into pci_get_port_hndl_isc to get a handle. The handle is needed to access the port through the pci_read_port_uint32_isc function.

PCI_PORT_HNDL phndl;
uint32_t pci_port_addr;
uint32_t data;

/*
 * get the io port address and mask off unwanted bottom
 * bits
 */
pci_read_cfg_uint32_isc(isc, mydriver_PORT_BASE_REG,
    &pci_port_addr);
pci_port_addr &= ~0x3;

/*
 * get the port handle
 */
if (pci_get_port_hndl_isc
   (isc, pci_port_addr, mydriver_PORT_BLOCK_SIZE, &phndl)) {
   /*
   * use it for as long as you want,
   * then return it when it is no longer needed
   */
   pci_read_port_uint32_isc(isc, phndl, mydriver_PORT_OFFSET,
                           &data);
   ...
   pci_unget_port_hndl_isc(
      isc, pci_port_addr, mydriver_PORT_BLOCK_SIZE, phndl);
}

SEE ALSO

pci_unget_port_hndl_isc(9F)
NAME

pci_read_cfg_uintN_isc - Read unsigned integer from a PCI configuration register.

SYNOPSIS

#include <sys/pci.h>

void pci_read_cfg_uint8_isc(
    struct isc_table_type *isc,
    int reg_num,
    uint8_t *data_read
);

pci_read_cfg_uint16_isc(
    struct isc_table_type *isc,
    int reg_num,
    uint16_t *data_read
);

pci_read_cfg_uint32_isc(
    struct isc_table_type *isc,
    int reg_num,
    uint32_t *data_read
);

PARAMETERS

isc Pointer to an ISC table associated with the device.
reg_num The offset of a PCI configuration register for the PCI device/function specified by isc. It can be a PCI_CS_* constant, defined in pci.h.
data_read A pointer to an 8-, 16-, or 32-bit location where the routine is to place the value.

DESCRIPTION

The pci_read_cfg_uintN_isc PCI functions read an 8-, 16-, or 32-bit unsigned integer from a PCI configuration register for a particular PCI device/function.

RETURN VALUES

The pci_read_cfg_uintN_isc routines do not return values.

CONSTRAINTS

None

EXAMPLE

#include <sys/pci.h>
static void
mydriver_set_io_master (struct isc_table_type * isc)
{
    unsigned short hwid;
    uint16_t old_cmdreg;

    PCI_PORT_HNDL ph;

    pci_read_cfg_uint16_isc(isc, PCI_CS_COMMAND, &old_cmdreg);
    pci_write_cfg_uint16_isc(isc, PCI_CS_COMMAND, old_cmdreg | PCI_CMD_IO_SPACE | PCI_CMD_BUS_MASTER);
    ...
}
SEE ALSO

pci_write_cfg_uintN_isc(9F)
NAME

pci_read_port_uintN_isc - Read little-endian data from an I/O port.

SYNOPSIS

#include <sys/pci.h>

void pci_read_port_uint8_isc(
    struct isc_table_type *isc,
    PCI_PORT_HNDL ph,
    uint32_t offset,
    uint8_t *data
);

pci_read_port_uint16_isc(
    struct isc_table_type *isc,
    PCI_PORT_HNDL ph,
    uint32_t offset,
    uint16_t *data
);

pci_read_port_uint32_isc(
    struct isc_table_type *isc,
    PCI_PORT_HNDL ph,
    uint32_t offset,
    uint32_t *data
);

PARAMETERS

isc     Pointer to an ISC table associated with the device.
ph      A port handle previously obtained with a call to pci_get_port_hndl_isc.
offset  An offset from ph.
data    A pointer to an 8-, 16-, 32-bit location where the routine is to place the value.

DESCRIPTION

The pci_read_port_uintN_isc PCI functions read 8-, 16-, or 32-bit little-endian data for the device/function specified by isc from the I/O port represented by the PCI port handle ph and offset offset. You will probably need to swap bytes if your driver will operate on 16, or 32 bit data.

RETURN VALUES

The pci_read_port_uintN_isc routines do not return values.

CONSTRAINTS

None

EXAMPLES

#include <sys/pci.h>

#define MY_IOMAP_BASE 0x10
#define MY_PORT_SIZE 0x100
#define MY_IDREG 0x0
#define MY_HWID 0x4850
static void
mydriver_memset(struct isc_table_type *isc)
{
    unsigned short hwid;
unsigned int port_addr;
uint16_t  old_cmdreg;
PCI_PORT_HNDL ph;

isc->mapped = NULL;
pci_read_cfg_uint16_isc(isc,PCI_CS_COMMAND,&old_cmdreg);
msg_printf("command reg = 0x%x\n",old_cmdreg);
pci_write_cfg_uint16_isc(isc, PCI_CS_COMMAND, old_cmdreg |
                         PCI_CMD_IO_SPACE | PCI_CMD_BUS_MASTER);
pci_read_cfg_uint32_isc(isc, MY_IOMAP_BASE, &port_addr);
port_addr &= ~3;
if (pci_get_port_hndl_isc(isc, port_addr,
                           MY_PORT_SIZE, &ph)) {
    pci_read_port_uint16_isc(isc,ph,MY_IDREG,&hwid);
    if ((hwid & MY_HWID) != MY_HWID) {
        return -1;
    }
} else {
    msg_printf("pci_get_port_hndl_isc() failed\n");
    return -1;
}
isc->mapped=(int)ph.hndl;
return 0;

SEE ALSO

pci_write_port_uintN_isc(9F)
NAME

pci_unget_port_hndl_isc - Delete a system defined handle for manipulating a range of PCI I/O ports.

SYNOPSIS

#include <sys/pci.h>

int pci_unget_port_hndl_isc(
    struct isc_table_type *isc,
    uint32_t pci_io_addr,
    uint32_t size,
    PCI_PORT_HNDL phndl
);

PARAMETERS

isc Pointer to an ISC table associated with the device.
pci_io_addr The address of a range of PCI I/O ports.
size The size of the PCI I/O ports.
phndl A handle obtained for these parameters by a previous call to pci_get_port_hndl_isc

DESCRIPTION

The pci_unget_port_hndl_isc PCI function deletes a system-defined handle for manipulating a range of PCI I/O ports.

RETURN VALUES

0 Failure — the handle could not be deleted by the system.
1 Success — the handle was deleted by the system.

CONSTRAINTS

None

EXAMPLES

None

SEE ALSO

pci_get_port_hndl_isc(9F)
pci_write_cfg_uintN_isc(9F)

NAME

pci_write_cfg_uintN_isc - Write unsigned integer to a PCI configuration register.

SYNOPSIS

#include <sys/pci.h>

void pci_write_cfg_uint8_isc(
    struct isc_table_type *isc,
    int reg_num,
    uint8_t data_write
);

cpyi_write_cfg_uint16_isc(
    struct isc_table_type *isc,
    int reg_num,
    uint16_t data_write
);

cpyi_write_cfg_uint32_isc(
    struct isc_table_type *isc,
    int reg_num,
    uint32_t data_write
);

PARAMETERS

isc Pointer to an ISC table associated with the device.
reg_num The number of a PCI configuration register for the PCI device/function specified by isc. It can be a PCI_CS_* constant, defined in pci.h.
data_write The 8-, 16-, or 32-bit value to be written.

DESCRIPTION

The pci_write_cfg_uintN_isc PCI functions write an 8-, 16-, or 32-bit unsigned integer to a PCI configuration register for a particular PCI device or function.

RETURN VALUES

The pci_write_cfg_uintN_isc routines do not return values.

CONSTRAINTS

None

SEE ALSO

pci_read_cfg_uintN_isc(9F)
pci_write_port_uintN_isc(9F)

NAME

pci_write_port_uintN_isc - Write little-endian data to an I/O port.

SYNOPSIS

#include <sys/pci.h>

void pci_write_port_uint8_isc(
    struct isc_table_type *isc,
    PCI_PORT_HNDL ph,
    uint32_t offset,
    uint8_t data
);

void pci_write_port_uint16_isc(
    struct isc_table_type *isc,
    PCI_PORT_HNDL ph,
    uint32_t offset,
    uint16_t data
);

void pci_write_port_uint32_isc(
    struct isc_table_type *isc,
    PCI_PORT_HNDL ph,
    uint32_t offset,
    uint32_t data
);

PARAMETERS

isc Pointer to an ISC table associated with the device.
ph A port handle previously obtained with a call to pci_get_port_hndl_isc.
offset An offset from ph.
data The 8-, 16-, or 32-bit value to be written.

DESCRIPTION

The pci_write_port_uintN_isc PCI functions write 8-, 16-, or 32-bit little-endian data for
the device/function specified by isc to the I/O port represented by the PCI port handle ph and
offset offset. You will probably need to swap bytes if your driver is operating on 16- or 32-bit
data.

RETURN VALUES

The pci_write_port_uintN_isc routines do not return values.

CONSTRAINTS

None

SEE ALSO

pci_read_port_uintN_isc(9F)
READ_REG_UINTn_ISC(9F)

NAME

READ_REG_UINTn_ISC - Read and byte swap data from a little-endian bus.

SYNOPSIS

```c
#include <sys/pci.h>

void READ_REG_UINT8_ISC(
    struct isc_table_type *isc,
    uint8_t *addr,
    uint8_t *data
);

#include <sys/pci.h>

void READ_REG_UINT16_ISC(
    struct isc_table_type *isc,
    uint16_t *addr,
    uint16_t *data
);

#include <sys/pci.h>

void READ_REG_UINT32_ISC(
    struct isc_table_type *isc,
    uint32_t *addr,
    uint32_t *data
);
```

PARAMETERS

isc Pointer to an ISC table associated with the driver.

addr A pointer to the address of the bus data. It must be one of:

- A virtual address mapped with map_mem_to_host.
- A mapped offset in the automatically mapped first-base-address register range contained in isc->if_reg_ptr.
- Only the first nonzero 32-bit-wide memory base register found can be mapped, starting in the range 0x10 and searching up through 0x24 (the six possible base address register locations in configuration space).
- If that base-register's size is in excess of 8 KB, it is not mapped and isc->if_reg_ptr is set to NULL. In this case, the driver must map the base register it wants to use.
- The address of a memory buffer shared between the driver and a little-endian bus master.

data A pointer to an 8-, 16-, or 32-bit location where the routine it to place the resultant data.

DESCRIPTION

The READ_REG_UINTn_ISC PCI services are macros that read and byte-swap data located at addr from a little-endian bus and place it in data.

If the PCI adapter that your card is running under has directly mapped the PCI memory space into driver-accessible system I/O space, you can improve the performance of READ_REG_UINTn_ISC if you define the PCI_LITTLE_ENDIAN_ONLY flag prior to including the pci.h header file. This causes READ_REG_UINTn_ISC to perform a simple byte swap instead of calling a function that tests byte ordering.
RETURN VALUES
The READ_REG_UINTn_ISC routines do not return values.

CONSTRAINTS
None

EXAMPLES
#include <sys/pci.h>
#define MY_REGISTER_OFFSET 0x40
/* the address of some register on my card */

uint8_t data8;
uint8_t *addr = isc->if_reg_ptr + MY_REGISTER_OFFSET;
    /* virtual address plus an offset */

    /* code accessing registers is expanded inline */
READ_REG_UINT8_ISC(isc, addr, &data8);

SEE ALSO
WRITE_REG_UINTn_ISC(9F)
WRITE_REG_UINTn_ISC(9F)

NAME

WRITE_REG_UINTn_ISC - Byte swap and write data to a little-endian bus.

SYNOPSIS

#include <sys/pci.h>

void WRITE_REG_UINT8_ISC(
    struct isc_table_type *isc,
    uint8_t *addr,
    uint8_t data
);

#include <sys/pci.h>

void WRITE_REG_UINT16_ISC(
    struct isc_table_type *isc,
    uint16_t *addr,
    uint16_t data
);

#include <sys/pci.h>

void WRITE_REG_UINT32_ISC(
    struct isc_table_type *isc,
    uint32_t *addr,
    uint32_t data
);

PARAMETERS

isc Pointer to an ISC table associated with the device.

addr A pointer to the output address. It must be one of the following:
- A virtual address mapped with map_mem_to_host.
- A mapped offset in the automatically mapped first-base-address register range contained in isc->if_reg_ptr.
  - Only the first nonzero 32-bit wide memory base register found can be mapped, starting in the range 0x10 and searching up through 0x24 (the six possible base address register locations in configuration space).
  - If that base-register's size is in excess of 8 KB, it is not mapped and isc->if_reg_ptr is set to NULL. In this case, the driver must map the base register it wants to use.
  - The address of a memory buffer shared between the driver and a little-endian bus master.

data The 8-, 16-, or 32-bit data to be written.

DESCRIPTION

The WRITE_REG_UINTn_ISC PCI services are macros that byte-swap and write data to a little-endian bus or to a host memory area shared by the driver and a little-endian bus master, located at addr.

If the PCI adapter that your card is running under has directly mapped the PCI memory space into driver-accessible system I/O space, you can improve the performance of WRITE_REG_UINTn_ISC if you define the PCI_LITTLE_ENDIAN_ONLY flag prior to including the pci.h header file. This causes WRITE_REG_UINTn_ISC to perform a simple byte swap instead of calling a function that tests byte ordering.
**RETURN VALUES**

The `WRITE_REG_UINTn_ISC` routines do not return values.

**CONSTRAINTS**

None

**EXAMPLES**

```c
#include <sys/pci.h>
#define MY_REGISTER_OFFSET 0x40

uint8_t  data8;
uint8_t  *addr = isc->if_reg_ptr + MY_REGISTER_OFFSET;
    /* virtual address plus an offset */

    /*
    * code accessing registers is expanded inline
    *
    */
    WRITE_REG_UINT8_ISC(isc, addr, &data8);

**SEE ALSO**

`READ_REG_UINTn_ISC(9F)`
NAME

PCI_ERRATA-1 -

Memory Coherency Issues

Certain combinations of WSIO mapping service calls can interact with PCI masters on C class and J class processors to create an inconsistent view of memory.

It is possible for prefetching of host memory by the PA hardware chipsets to result in a PCI master reading stale data, even though the proper dmaync calls have been made. The problem does NOT occur if:

- The PCI master does normal Memory Read transactions; the master does not master Memory Read Multiple (MRM) or Memory Read Line (MRL) transactions.
- The mapping is done with wsio_map with the IO_NO_SEQ and IO_SAFE flags set, regardless of the type of transactions the PCI master uses.
- The mapping is done with wsio_fastmap and the PCI master does NOT use MRM or MRL transactions.

DETAILS

There are two hardware prefetch buffers in the PA hardware chipset between memory and any PCI device. One is in the system’s PCI bridge chip, and the other is system’s GSC I/O bridge chip (which connects to the PA side of the PCI bridge chip). The I/O bridge chip has a cache line size prefetch buffer for each I/O (GSC) slot.

For the following discussion assume that an IO TLB was mapped using either wsio_fastmap or wsio_map without IO_NO_SEQ and IO_SAFE flag bits set.

When a PCI bus master runs an MRM or MRL transaction, the following events happen:

1. The PCI bridge chip requests a cache line (8 words), starting at the PCI master requested start address, from the I/O bridge chip. Because the PCI master is running an MRM or MRL transaction, this request is made with a prefetch hint enabled for the I/O bridge chip. In addition, it requests subsequent cache lines from the I/O bridge chip, with the exact number of extra lines dependent upon whether an MRM or MRL PCI transaction is in progress, and whether or not the end of a physical page is near (the PCI bridge chip will not prefetch past the end of a page).
2. The I/O bridge chip, for each cache line requested, fills the request immediately from its own prefetch buffer if the requested line resides there, or gets the cache line from processor memory. It then immediately prefetches the next line from processor memory into its prefetch buffer.

Two problems exist. The first case is when the PCI bridge chip has requested the cache line at the end of a physical page (note that this does not imply that the PCI device, itself, has requested the cache line at the end of the page); for example, 0x0fe0. The PCI bridge chip, incorrectly requests this cache line with the prefetch hint enabled. The I/O bridge chip, to avoid fetching onto a possible non-existent page, but needing to do something with the prefetch hint enabled, prefetches the first cache line of the page (0x0000). In the example case, stale data can be read if the next request from the PCI master is for address 0x0000, which has just been incorrectly prefetched.

This case is fairly easy to hit. A driver might have control information consisting of a list of multiple structures that just fill a physical page. If the PCI bridge reads the last cache line of the page, followed by the driver re-writing the list, doing a dma_sync, and then directing the PCI master to re-read the list, the PCI master will read stale data in the first cache line.

The second case is where a driver has two adjacent data structures on the same page. The PCI master reads from the first data structure. The PCI bridge chip and the I/O bridge chip have
prefetched such that the I/O bridge chip has a cache line in its prefetch buffer that actually resides in the second data structure. If the PCI master then reads that particular address, it may have stale data (depending upon the sequence the driver follows in updating it vis-a-vis the PCI master’s access).

In both cases, if the mapping is done using wsio_map with IO_NO_SEQ and IO_SAFE flag bits set, no problem exists (because the I/O bridge chip ignores the prefetch hint when the IO TLB is set up by WSIO mapping services with this mapping).

Note that the IO_NO_SEQ and IO_SAFE flag bits will degrade MRM and MRL performance by about a factor of two for the page(s) in this type of mapping. If it is absolutely necessary, for performance reasons, to use wsio_fastmap or wsio_map without the IO_NO_SEQ and IO_SAFE flag bits set the coherency problem can be prevented by having the PCI master read a different address which will reset the I/O bridge chip’s prefetch buffer. A read of any address using MRM, MRL or a normal read transaction by the PCI master will accomplish this, so you might have the PCI master re-read the previous cache line and then throw it away.

**SEE ALSO**

wsio_map(9F), wsio_fastmap(9F)
PCI TRANSACTION ORDERING

Because of the interaction between the host bus, PCI bridge chips, and the PCI bus, in certain situations, the Producer Consumer model requirements defined in the PCI 2.1 Specification may not be met. For more detailed information, see the discussion on Transaction Ordering in the PCI Chapter of the *HP-UX 11i v3 Driver Development Guide*.
PCI.CONFIGURATION CYCLE RETRY PROBLEM

The system's PCI bridge chip holds IRDY too long on config write retry. This problem has only been seen in simulation with some revisions of the PCI bridge chip used on the B1000, C3000, J5000, and the N-Class servers.

This problem occurs when:

- A device retries a configuration cycle.
- The device asserts DEVSEL and STOP during the same cycle.
- That cycle is not the one immediately following the address cycle then the PCI bridge chip will ignore the RETRY, believe that the card never asserted DEVSEL response, which will cause a master abort.

If the card asserts DEVSEL for one or more cycles before it asserts STOP, the problem does not occur.

In PCI bridge chips exhibiting this behavior, the most likely result is an HPMC or panic.
4 Network Device Driver Reference Pages

This chapter contains manual reference pages for the data structures, kernel and user space support routines, and macros essential for HP-UX network interface drivers.
DL_DISABMULTI_REQ(9G)

NAME

DL_DISABMULTI_REQ - Request to disable multicast address.

SYNOPSIS

#include <sys/dlpi.h>

DESCRIPTION

This primitive is sent to drivers using the driver_controlp routine to disable a multicast address. The multicast address information is present in the dl_disabmulti_req_t structure.

The DL_DISABMULTI_REQ primitive described in this manpage may be different from the definition in DLPI Programmer’s Guide. The information provided here is applicable only to tightly coupled drivers.

STRUCTURE MEMBERS

typedef struct {
    uint32_t     dl_primitive;
    uint32_t     dl_addr_length;
    uint32_t     dl_addr_offset;
} dl_disabmulti_req_t;

dl_primitive   Set to DL_DISABMULTI_REQ.
dl_addr_length Length of the multicast address in bytes.
dl_addr_offset Offset from the beginning of the message block (MBLK->b_rptr) where the multicast address begins.

CONSTRAINTS

HP-DLPI will handle MAC and multicast addresses in canonical format for Ethernet drivers. HP-DLPI will always succeed the DL_DISABMULTI_REQ request to the user/application, even if driver returned an error. If the driver fails the request for some reason, it will go out-of-sync with HP-DLPI and the driver can attempt to reprogram the multicast addresses (by getting them from HP-DLPI using a property) at a convenient time. Failure to do so, will result in performance degradation.

SEE ALSO

driver_controlp(9F)
NAME

DL_ENABMULTI_REQ - Request to enable multicast address.

SYNOPSIS

#include <sys/dlpi.h>

DESCRIPTION

This primitive is sent to drivers using the driver_controlp routine to enable a multicast address. The multicast address information is present in the dl_enabmulti_req_t structure.

The DL_ENABMULTI_REQ primitive described in this manpage may be different from the definition in DLPI Programmer’s Guide. The information provider here is applicable only to tightly coupled drivers.

STRUCTURE MEMBERS

typedef struct {
    uint32_t  dl_primitive;
    uint32_t  dl_addr_length;
    uint32_t  dl_addr_offset;
} dl_enabmulti_req_t;

dl_primitive                      Set to DL_ENABMULTI_REQ
dl_addr_length                    Length of multicast address in bytes.
dl_addr_offset                    Offset from the beginning of the message block (MBLK->b_rptr of the message block) where the multicast address begins.

CONSTRAINTS

HP-DLPI will handle MAC and multicast addresses in canonical format for Ethernet drivers.

HP-DLPI will send DL_ENABMULTI_REQ only for unique multicast addresses (if multiple streams enable same multicast address on the driver instance, only one DL_ENABMULTI_REQ will be sent).

HP-DLPI will automatically convert the DL_ENABMULTI_REQ to a multicast level, promiscuous-on request if the number of unique multicast addresses enabled on the interface exceeds the limit specified by the driver in dhc_max_mcast element of dl_hp_create_info_t. This will be done only if the driver instance is not already in multicast or higher promiscuous level.

SEE ALSO

driver_controlp(9F)
NAME

DL_GET_STATISTICS_REQ - Request to get the physical interface statistics.

SYNOPSIS

#include <sys/dlpi.h>

DESCRIPTION

This primitive is sent to the drivers using the driver_controlp routine to get the physical interface statistics. The associated structure for this request is dl_get_statistics_req_t structure.

The DL_GET_STATISTICS_REQ primitive described in this manpage may be different from the definition in DLPI Programmer’s Guide. The information provided here is applicable only to tightly coupled drivers.

STRUCTURE MEMBERS

typedef struct {
    uint32_t dl_primitive;
} dl_get_statistics_req_t;

dl_primitive Set to DL_GET_STATISTICS_REQ.

CONSTRAINTS

The second MBLK (linked by b_cont) will contain space to hold the statistics of size Ext_mib_t.

The second MBLK will be empty when it is passed to the driver from HP-DLPI (b_rptr == b_wpwp). After copying the statistics information to the second MBLK, the drivers must move the b_wptr to the size that was written to the second MBLK (in this case by the size of the Ext_mib_t). The driver can cast the b_rptr to be Ext_mib_t *, if it has to change some fields.

Drivers must not modify the b_rptr.

Even if driver supports only a subset of the statistics groups in Ext_mib_t structure, it must copy the complete structure.

Ext_mib_t is defined in <sys/mib.h>

SEE ALSO

driver_controlp(9F)
NAME

dl_hp_autoneg_sense_t - Auto Negotiation or Auto Sense value of the network interface.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

This enumeration contains the values of Auto Negotiation or Auto Sense. This is used in
dl_hp_op_param_t structure to set the driver instance auto-negotiation or auto-sense values.

STRUCTURE MEMBERS

typedef enum dl_hp_autoneg_sense {
    DL_HP_AUTONEG_SENSE_ON   = 0x1,
    DL_HP_AUTONEG_SENSE_OFF  = 0x2
} dl_hp_autoneg_sense_t;

DL_HP_AUTONEG_SENSE_ON       Auto Negotiation or Auto Sense is ON.
DL_HP_AUTONEG_SENSE_OFF      Auto Negotiation or Auto Sense is OFF.

SEE ALSO

dl_hp_op_param_t(9S)
dl_hp_cmd_info_t(9S)

NAME
dl_hp_cmd_info_t—Structure containing information about one attached stream on an interface.

SYNOPSIS
#include <sio/dlpi_ext.h>

DESCRIPTION
This structure defines the information returned for a stream that is attached in the interface to
be analyzed. Note the command name and pid might be out of date if the stream is persistently
linked to a STREAMS multiplexer (mux) module by the I_LINK ioctl. For example, the ifconfig
command, which plumbs the IP stream, exits before the usage information is returned.

STRUCTURE MEMBERS

typedef union dl_hp_cmd_info {
    int8_t dl_cmd_name[MAXCOMLEN+1];
    pid_t dl_pid;
    uint32_t dl_sap_length;
    uint8_t dl_sap[MAX_SAP_LEN];
    uint32_t dl_reserved1;
    uint32_t dl_reserved2;
} dl_hp_cmd_info_t;

dl_cmd_name Specifies the command name of the application that opened the stream.
dl_pid The PID of the process that opened the stream. Note the process may be
     exited although the stream is persistent in the kernel. In this case, ignore
     the PID.

dl_sap_length The length of the dl_sap field. A length of 0 indicates that the stream is
     not attached. The possible values are:
     1 DL_HP_LAN_SAP type, dl_sap[0] contains the SAP value.
     2 DL_HP_LAN_Type type, dl_sap[0-1] contains the protocol value.
     3 DL_HP_LAN_CANON type, dl_sap[0] is either IEEESAP_HP or
        IEEESAP_NM, and dl_sap[1-2] contains the extended sap value.
     6 DL_HP_LAN_SNAP type, dl_sap[0]=0xAA, dl_sap[1-3] contains
        the organization id or 0 for IP, dl_sap[4-5] contains the protocol
        value.

dl_sap SAP value. see dl_sap_length for information.
dl_reserved1 Reserved for future use. Must be set to 0.
dl_reserved2 Reserved for future use. Must be set to 0.

SEE ALSO
dl_hp_info_t(9S), dl_hp_info_type_t(9S), dl_hp_usage_info_t(9S)
NAME
dl_hp_create_info_t - Structure to be passed during DL_HP_OP_CREATE operation.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
This data type used by tightly coupled or loosely coupled driver to register their instance with
HP-DLPI. HP-DLPI will return the per-instance handle that identifies the HP-DLPI instance in
later calls to HP-DLPI interfaces.

STRUCTURE MEMBERS

typedef struct dl_hp_create_info {
    uint32_t dhc_version;
    uint32_t dhc_instance_num;
    uint32_t dhc_ppa;
    dl_hp_mac_type_t dhc_mac_type;
    uint32_t dhc_mac_addr_len;
    uint32_t dhc_nm_id;
    uint32_t dhc_mtu;
    uint32_t dhc_major_num;
    uint32_t dhc_max_mcast;
    uint32_t dhc_max_vlan;
    dl_hp_drv_features_one_t dhc_features_one;
    uint32_t dhc_features_two;
    uint32_t dhc_features_three;
    dl_hp_encaps_type_t dhc_encaps;
    uint8_t dhc_mac_addr[DL_HP_MAX_MAC_ADDR_LEN];
    char dhc_hw_path[DL_HP_MAX_HDW_PATH_LEN];
    char dhc_drv_name[DL_HP_MAX_DRIVER_NAME_LEN];
    char dhc_arpmod_name[DL_HP_MAX_ARPMOD_NAME_LEN];
    uint32_t (*dhc_outputp)(void *drv_output_hdlp,
                           mblk_t *mblkp,
                           dl_hp_pkt_type_t pkt_type,
                           void *rsvdp);
    uint32_t (*dhc_build_hdrp)(void *drv_output_hdlp,
                              dl_hp_src_llc_info_t *src_infop,
                              dl_hp_dest_llc_info_t *dset_infop,
                              dl_hp_llc_info_t *hdr_infop,
                              void *rsvd1p,
                              void *rsvd2p);
    void (*dhc_controlp)(void *drv_control_hdlp,
                         int32_t cmd,
                         mblk_t *mblkp,
                         void *rsvdp);
    int32_t (*dhc_eventp)(void *drv_event_hdlp,
                          dl_hp_drv_event_type_t event_type,
                          void *event_infop,
                          void *rsvdp);
    int32_t (*dhc_drv_propp)(void *drv_prop_hdlp,
                             uint32_t opcode,
                             uint32_t prop_name,
                             void *valuep,
                             void *rsvd1p,
                             void *rsvd2p);
    void *dhc_output_hdlp;
    void *dhc_control_hdlp;
    void *dhc_event_hdlp;
    void *dhc_mod_hdlp;

} dl_hp_create_info_t;
void *dhc_dlpi_hdlp;
dl_hp_drv_features_one_t dhc_features_one_cap;
uint32_t dhc_features_two_cap;
uint32_t dhc_features_three_cap;
void *dhc_reserved[3];
} dl_hp_create_info_t;

dhc_arpmod_name
From driver to HP-DLPI.
Character string specifying the ARP (STREAMS) module
name for this driver, if one is required.
The name must be less than or equal to
DL_HP_MAX_ARPMOD_NAME_LEN characters and must be
NULL terminated. The DL_HP_MAX_ARPMOD_NAME_LEN
constant is defined as 64.
Set by drivers only during registration of the interface.

dhc_build_hdrp
From driver to HP-DLPI.
The driver routine to build the MAC/LLC header in the
outbound path. See driver_build_hdrp(9F) for details.
Set by physical drivers only during registration of the driver
instance.
The driver can set this value to NULL if it does not have a
build header routine. In this case, HP-DLPI uses its internal
build header routine for Ethernet drivers.

dhc_control_hdlp
From driver to HP-DLPI.
The driver specified opaque (to HP-DLPI) handle that is
passed to the driver control request processing routine.

dhc_controlp
From driver to HP-DLPI.
The driver routine for all control requests. See
driver_controlp(9F) for more information.
Set by drivers only during registration of the driver instance.

dhc_dlpi_hdlp
From HP-DLPI to the driver.
Opaque handle that drivers must use in future call to HP-DLPI
interfaces.

dhc_drv_name
From driver to HP-DLPI.
Driver name as a (null terminated) character string. The name
must be less than or equal to DL_HP_MAX_DRIVER_NAME_LEN
characters and must be NULL terminated. The
DL_HP_MAX_DRIVER_NAME_LEN constant is defined as 64.
Set by drivers only during registration of the interface.

dhc_drv_propp
From driver to HP-DLPI.
Driver's property exchange routine. DLPI does not call this
driver interface.
This information is optional and the field must be set to NULL
if it is not implemented.
dhc_encaps From driver to HP-DLPI.
Specifies the encapsulations supported by the drivers. For supported encapsulation types, refer to dl_hp_encaps_type_t.
Set by physical drivers only during registration of the interface.
The field is displayed by the lanscan command.

dhc_event_hdlp From driver to HP-DLPI.
The driver specified opaque (to HP-DLPI) handle that is passed to the driver event handler.

dhc_eventp From driver to HP-DLPI.
The driver event handler to handle events defined by the DLS provider. See driver_eventp(9F) for more information.
Set by drivers only during registration of the driver instance.

dhc_features_one From driver to HP-DLPI.
See dl_hp_features_one_t(9S) for the features supported and the explanation of each one of the features.
Set by drivers during registration and can be changed using DLPI_PROP_FEATURES_ONE.

dhc_features_one_cap From driver to HP-DLPI.
The feature capabilities of the interface. The dhc_features_one field is a subset of the features set in this field because some features can be turned off even though the driver is capable of supporting the feature. This field is supported from version 3 and above.
See dl_hp_features_one_t(9S) for the features supported and the explanation of each one of the features.

dhc_features_two From driver to HP-DLPI.
Reserved for future use. Set to 0.

dhc_features_two_cap From driver to HP-DLPI.
Reserved for future use. Set to 0. This is supported from version 3 and above.

dhc_features_three From driver to HP-DLPI.
Reserved for future use. Set to 0.

dhc_features_three_cap From driver to HP-DLPI.
Reserved for future use. Set to 0. This is supported from version 3 and above. If drivers set dhc_version to 2, HP-DLPI will not expect this field.

dhc_hdw_path From driver to HP-DLPI.
The hardware path of the physical interface as a (null terminated) character string. The length of the hardware path could be a maximum of DL_HP_MAX_HDW_PATH_LEN bytes, including NULL termination. The DL_HP_MAX_HDW_PATH_LEN constant is defined as 100.
Logical drivers can use the hardware path to specify a unique string for each instance of the logical driver.
Set by drivers only during registration of the interface.
The field is displayed by the `lanscan` command.

**dhc_instance_num**
From driver to HP-DLPI.
An identifier that maps one-to-one with a physical interface belonging to a particular class. Set by drivers only during registration of the interface. The field is displayed by the `lanscan` command.

**dhc_mac_addr_len**
From driver to HP-DLPI.
Length in bytes of the MAC address. Set by physical drivers only during registration of the interface.

**dhc_mac_addr**
From driver to HP-DLPI
The MAC address of the card, assumed to be `dhc_mac_addr_len` bytes in length and a maximum of `DL_HP_MAX_MAC_ADDR_LEN` bytes in length. The `DL_HP_MAX_MAC_ADDR_LEN` constant is defined as 32.
The MAC address must be in canonical format for Ethernet and FDDI drivers and in wire format for TR drivers.
Most drivers may not be able to provide the MAC address at the time of registering the instance. Hence, drivers need to explicitly set the MAC address (via `DLPI_PROP_MAC_ADDRESS` property) after it is available.
The drivers which are unable to set the MAC addresses during create must set this value to 0.

**dhc_mac_type**
From driver to HP-DLPI.
Set by drivers only during registration of the interface. The field is displayed by the `lanscan` command.

**dhc_major_num**
From driver to HP-DLPI.
This field is valid only for loosely coupled drivers. For tightly coupled drivers HP-DLPI sets this value to the HP-DLPI major number.
This field must be set to the major number of the DLS module used by the driver. Set by drivers only during registration of the interface.
Field is displayed by the `lanscan` command.

**dhc_max_mcast**
From driver to HP-DLPI.
The driver specified maximum number of unique multicast addresses supported by the driver/NIC in the range 1-0x7FFFFFFF.
Set by physical drivers only during registration of the driver instance and cannot be modified thereafter.
Drivers that do not have any such limits (such as those that use imperfect filtering) must set this field to 0xFFFFFFFF.
dhc_max_vlan  From driver to HP-DLPI.  
The driver specified maximum number of VLANs supported by the driver or NIC in the range 0-4094.  
Drivers can set this only during registration of the driver instance; it cannot be modified at a later time.

dhc_mtu  From driver to HP-DLPI.  
The default MTU for the link in bytes. The minimum MTU must be 10 and the minimum recommended MTU is 256.  
Set by drivers during registration of the interface. Can be modified anytime later (via the DLPI_PROP_OP_PARAM property)

dhc_mod_hdlp  From driver to HP-DLPI.  
Reserved for future use. Must be set to NULL.

dhc_nm_id  From driver to HP-DLPI.  
The network management ID. Set by loosely coupled drivers during registration of the interface and can be updated later via the DLPI_PROP_NMID property.  
This field is ignored for tightly coupled drivers as the HP-DLPI provider sets this property for an interface instance and provides it to drivers as a “get” property (DLPI_PROP_NMID). The DLPI_PROP_NMID property must be called by tightly coupled drivers if the NMID needs to be switched (atomically) for high availability applications (especially with IPv6). Tightly coupled drivers will be notified of the new NMID via the event handler interface (DL_HP_DRV_EVENT_CHANGE_NMID event).  
Loosely coupled drivers must provide this value during registration and if the NMID changes during the course of operation, they must use DLPI_PROP_NMID “set” property to update the value in DLPI.

dhc_output_hdlp  From driver to HP-DLPI.  
The driver specified opaque (to HP-DLPI) handle that is passed to the outbound routines.

dhc_outputp  From driver to HP-DLPI.  
The driver outbound routine. See driver_outputp(9F) for information.  
Set by drivers only during registration of the driver instance.

dhc_ppa  From driver to HP-DLPI.  
PPA of the interface. Valid only for loosely coupled drivers. For tightly coupled drivers, the dhc_instance_num will be used as PPA.

dhc_reserved  Reserved for future use. Must be set to 0.

dhc_version  From driver to HP-DLPI.  
HP-DLPI Version that the driver can work with. Set by drivers only during registration of the interface.
SEE ALSO

dlpi_propp(9F), driver_contolp(9F), driver_eventp(9F), dl_hp_encaps_type_t(9S),
dl_hp_drv_features_one_t(9S), lanscan(1M), driver_build_hdrp(9F)
NAME

DL_HP_CREATE_VLAN_REQ - Request to create a VLAN interface over an interface.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

This primitive can be sent by user space to the DLS provider HP-DLPI. This primitive will be sent to drivers by HP_DLPI using the driver_controlp routine to create a VLAN on the interface. The associated structure for this request is dl_hp_create_vlan_req_t structure.

STRUCTURE MEMBERS

typedef struct dl_hp_create_vlan_req {
    uint32_t   dl_primitive;
    uint32_t   dl_ppa;
    req_type_t dl_req_type;
    uint16_t   dl_vlanid;
    uint16_t   dl_tos;
    uint32_t   dl_priority;
    char       dl_vlan_name[DL_HP_MAX_VLAN_NAME_LEN];
    pri_override_t dl_pri_override;
    tos_override_t dl_tos_override;
    uint32_t   dl_vppa;
    uint32_t   dl_reserved1[2];
    uint32_t   dl_reserved2[2];
} dl_hp_create_vlan_req_t;

dl_primitive
    Set to DL_HP_CREATE_VLAN_REQ.

dl_ppa
    Physical Point of Attachment (PPA) number on which to create the VLAN interface.

dl_req_type
    This is a enumeration that indicates which fields were set by the DLS user when a create or modify VLAN request is sent. See req_type_t for more information.

dl_vlanid
    VLAN ID for the VLAN. Valid values are 0 to 4094, inclusive.

dl_tos
    User assigned inbound TOS value for the VLAN. Valid values are 0 to 255, inclusive. The default setting is 0.

dl_priority
    User assigned 802.1p outbound priority for the VLAN. Valid values are 0 to 7, inclusive. The default setting is 0.

dl_vlan_name
    Name of the VLAN. If not set, DLPI initializes the VLAN name to NULL.

dl_pri_override
    Outbound 802.1p priority override level (determines 802.1p priority setting of outbound frames). See dl_priOverride_t.

dl_tos_override
    Inbound ToS override level (determines the ToS setting of inbound packets). Applies only to IP. See dl_tos_override_t.

dl_vppa
    The VPPA number requested by the user.

dl_reserved1[2]
    Reserved for future use.

dl_reserved2[2]
    Reserved for future use.

See Also

driver_controlp(9F), DL_HP_DELETE_VLAN_REQ(9G), DL_HP_MODIFY_VLAN_REQ(9G)
**NAME**

DL_HP_DELETE_VLAN_REQ - Request to delete a VLAN interface over an interface.

**SYNOPSIS**

```c
#include <sys/dlpi_ext.h>
```

**DESCRIPTION**

This primitive can be sent by user space to the DLS provider, HP-DLPI, and from HP-DLPI to drivers using the `driver_controlp` routine to delete a VLAN on the interface. This message consists of a M_PROTO message block containing a structure of type `dl_hp_delete_vlan_req_t`.

This primitive applicable between HP-DLPI and the driver and between a DLPI user and HP-DLPI.

**STRUCTURE MEMBERS**

```c
typedef dl_hp_delete_vlan_req {
    uint32_t       dl_primitive;
    uint32_t       dl_vppa;
} dl_hp_delete_vlan_req_t;
```

- `dl_primitive` Set to DL_HP_DELETE_VLAN_REQ.
- `dl_vppa` The VPPA number of the VLAN interface to be deleted.

**SEE ALSO**

`driver_controlp(9F), DL_HP_CREATE_VLAN_REQ(9G), DL_HP_MODIFY_VLAN_REQ(9G)`.
NAME

DL_HP_MODIFY_VLAN_REQ - Request to modify a VLAN interface's attributes.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

This primitive can be sent by user space to the DLS provider, HP-DLPI, and from HP-DLPI to
drivers using the driver_controlp routine to modify a VLAN's attributes. This message
consists of an M_PROTO message block containing a structure of type
dl_hp_modify_vlan_req_t.

This primitive applicable between HP-DLPI and the driver and between a DLPI user and HP-DLPI.

STRUCTURE MEMBERS

typedef struct dl_hp_modify_vlan_req {
    uint32_t       dl_primitive;
    uint32_t       dl_vppa;
    req_type_t     dl_req_type;
    uint16_t       dl_vlanid;
    uint16_t       dl_tos;
    uint32_t       dl_priority;
    pri_override_t dl_pri_override;
    tos_override_t dl_tos_override;
    char           dl_vlan_name[DL_HP_MAX_VLAN_NAME_LEN];
    uint32_t       private;
    uint32_t       dl_reserved1[2];
    uint32_t       dl_reserved2[2];
} dl_hp_modify_vlan_req_t;

dl_primitive  Set to DL_HP_MODIFY_VLAN_REQ.
dl_vppa        The Physical Point of Attachment (PPA) of the VLAN interface to be
                modified.
dl_req_type    This is a enumeration that indicates which fields were set by the DLS
                user when a modify VLAN request is sent. See req_type_t for more
                information.
dl_vlanid      New VLAN ID for the VLAN. Valid values are 0 to 4094, inclusive.
dl_tos         New inbound TOS value for the VLAN. Valid values are 0 to 255,
                inclusive. The priority that is actually set in the outbound packet is
determined by HP-DLPI after looking at the dhvi_tos_override value. See
dl_tos_override_t.
dl_priority    New 802.1p outbound priority for the VLAN. Valid values are 0 to 7,
                inclusive. The TOS that is actually set in the inbound packet is
determined by HP-DLPI after looking at the dhvi_pri_override value. See
dl_pri_override_t.
dl_pri_override Outbound 802.1p priority override level (determines 802.1p priority
                setting of outbound frames). See dl_pri_override_t.
dl_tos_override Inbound ToS override level (determines the ToS setting of inbound
                packets). Applies only to IP. See dl_tos_override_t.
dl_vlan_name   New name of the VLAN.
dl_private
  Reserved for internal use.

dl_reserved1[2]
  Reserved for future use.

dl_reserved2[2]
  Reserved for future use.

**SEE ALSO**

driver_controlp(9F), DL_HP_CREATE_VLAN_REQ(9G), DL_HP_MODIFY_VLAN_REQ(9G)
NAME
dl_hp_dest_llc_info_t - Exchange destination information during LLC build header routine.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
This structure is used to pass destination information to the LLC build header routine from the DLS provider before the call to the driver handle is made.

STRUCTURE MEMBERS

typedef struct dl_hp_dest_llc_info {
    uint32_t ddli_dest_addr_length;
    uint8_t ddli_dest_addr[DL_HP_MAX_DLSAP_LEN];
    uint64_t ddli_reserved1[2];
    uint64_t ddli_reserved2[2];
} dl_hp_dest_llc_info_t;

ddli_dlsap_addr
Identifies the destination DLSAP address. See the DLPI Programmer’s Guide for more information on supported DLSAP formats.

ddli_dlsap_addr_length
Identifies the DLSAP address length in bytes.
ddli_reserved1
Reserved for future use. It must be set to 0.
ddli_reserved2
Reserved for future use. It must be set to 0.

SEE ALSO
dl_hp_drv_event_type_t(9S)

NAME
dl_hp_drv_event_type_t - Tightly coupled driver events

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
This structure provides an enumeration for events passed from HP-DLPI to a tightly coupled
driver interface. Of these events, only DL_HP_DRV_EVENT_CHANGE_NMID is sent to physical
driver modules. The remaining events are sent only to the logical drivers.

STRUCTURE MEMBERS
typedef enum dl_hp_drv_event_type {
    DL_HP_DRV_EVENT_INTERFACE_UP = 0x1,
    DL_HP_DRV_EVENT_INTERFACE_DOWN,
    DL_HP_DRV_EVENT_PORT_ADD,
    DL_HP_DRV_EVENT_PORT_DELETE,
    DL_HP_DRV_EVENT_CHANGE_NMID,
    DL_HP_DRV_EVENT_OLD,
    DL_HP_DRV_EVENT_IFISHWPATH,
    DL_HP_DRV_EVENT_CLEAR_STATS
} dl_hp_drv_event_type_t;

DL_HP_DRV_EVENT_INTERFACE_UP       Reserved for internal use.
DL_HP_DRV_EVENT_INTERFACE_DOWN     Reserved for internal use.
DL_HP_DRV_EVENT_PORT_ADD           Reserved for internal use.
DL_HP_DRV_EVENT_PORT_DELETE        Reserved for internal use.
DL_HP_DRV_EVENT_CHANGE_NMID        When the network management ID (NMID) for a driver
instance is changed by an upper layer module, HP-DLPI
informs the driver of the new NMID. The driver must then
take appropriate action.

This includes setting the appropriate field in the driver
(MIB) structures to reflect the changed NMID.

The event_infop parameter is of type (uint32_t *) that
references the new NMID of the interface.

Some NICs may require the NIC hardware to be
reprogrammed to the new value.

Note for logical drivers:
When the NMID of a logical driver instance is changed.
The logical instance private event handle,
dhc_event_hdlp, is passed in the driver_hdlp parameter
to the instance level event handler function
(dhc_eventp). These handles are specified during
registration of the logical driver instance.

When the NMID of a physical driver instance acquired by
a logical module is changed, the driver_hdlp parameter to
the module level event handler (driver_eventp) is the
dhma_mod_privatep field of the
dl_hp_mod_acquire_t data type passed when acquiring the physical driver instance.

The event_info parameter is a pointer to the dl_hp_drv_notify_nmid_change_t data type that has the PPA and the new NMID of the driver instance.

DL_HP_DRV_EVENT_OLD

During online deletion (OLD) of a physical interface, the driver sends this event to request DLPI to quiesce all access to the driver instance through DLPI. DLPI invokes the callback passed in the event to notify the driver as soon as all ongoing accesses have completed and DLPI can guarantee no future accesses to the driver instance.

DL_HP_DRV_EVENT_IFISHWPATH

Reserved for internal use.

DL_HP_DRV_EVENT_CLEAR_STATS

Reserved for internal use.

SEE ALSO
dl_hp_mod_acquire_t(9S), dl_hp_drv_notify_link_t(9S), driver_event_handlerp(9F),
dl_hp_drv_notify_nmid_change_t(9S)
NAME

dl_hp_drv_features_one_t - Driver features.

SYNOPSIS

#include <sio/dlpi_drv.h>

DESCRIPTION

Enumeration of driver features. These are bit-wise flags. Each feature consumes a bit in the
dhc_drv_features_one field passed to dlpi_propp. Drivers may need to use a combination of
these bits to specify their capabilities.

STRUCTURE MEMBERS

typedef enum dl_hp_drv_features_one {
    DL_HP_DRV_HP_DLS,
    DL_HP_DRV_IPV4CKO_IN,
    DL_HP_DRV_IPV4CKO_OUT,
    DL_HP_DRV_IPV6CKO_IN,
    DL_HP_DRV_IPV6CKO_OUT,
    DL_HP_DRV_COW,
    DL_HP_DRV_LFP,
    DL_HP_DRV_LNP,
    DL_HP_DRV_NO_SRC_ROUTING,
    DL_HP_DRV_NO_FASTPATH,
    DL_HP_DRV_HP_APA,
    DL_HP_DRV_VLAN,
    DL_HP_DRV_VTO,
    DL_HP_DRV_VI,
    DL_HP_DRV_PHYSICAL,
    DL_HP_DRV_LINKAGG,
    DL_HP_DRV_XPORT_OFFLOAD,
    DL_HP_DRV_LAN_CLASS,
    DL_HP_DRV_NOMULTIFRAGCKO,
    DL_HP_DRV_ILAN,
    DL_HP_DRV_TRAIN,
    DL_HP_DRV_64BIT_MIB,
    DL_HP_DRV_MGMT_INSTANCE
    DL_HP_DRV_PORT
    DL_HP_DRV_IPV4TCPSEG
    DL_HP_DRV_IPV6TCPSEG
    DL_HP_DRV_USAGE_INFO
    DL_HP_DRV_SUPP_CRA
} dl_hp_drv_features_one_t;

DL_HP_DRV_HP_DLS

Setting this feature bit says that the driver is a tightly
coupled driver. Loosely coupled drivers must not set this
feature bit. Loosely coupled drivers are limited to using
dlpi_propp to:
• Register the interface.
• Change certain fields like MAC address, hardware
  state for display via lanscan.
• Set operational parameter MTU.
Other HP-DLPI interfaces (inbound routine, event handler, and so on) must not be used by loosely coupled drivers. Except DL_HP_DRV_LAN_CLASS, and DL_HP_DRV_64BIT_MIB features, no other feature bits must be set.

HP-DLPI will not allow streams to attach to the interfaces that do not have this features set.

**DL_HP_DRV_IPV4CKO_IN**

Driver supports inbound IPv4 CKO. The drivers setting this feature bit must get the inbound OOP template from HP-DLPI using the DLPI_PROP_OOP_INFOP property. This feature may be enabled/disabled after the initialization. Any change in feature must trigger an associated link DOWN and UP events. The change in this feature must also be followed by a DLPI_PROP_OOP_INFOP property. See the *HP-UX 11i v3 Driver Development Guide* for more information.

**DL_HP_DRV_IPV4CKO_OUT**

Driver supports outbound IPv4 CKO. The drivers setting this feature bit must get the inbound OOP template from HP-DLPI using the DLPI_PROP_OOP_INFOP property. This feature may be enabled/disabled after the initialization. Any change in feature must trigger an associated link DOWN and UP events. The change in this feature must also be followed by a DLPI_PROP_OOP_INFOP property. See the *HP-UX 11i v3 Driver Development Guide* for more information.

**DL_HP_DRV_IPV6CKO_IN**

For drivers that support inbound IPv6 CKO. The drivers setting this feature bit must get the inbound OOP template from HP-DLPI using the DLPI_PROP_OOP_INFOP property. This feature may be enabled/disabled after the initialization. Any change in feature must trigger an associated link DOWN and UP events. The change in this feature must also be followed by a DLPI_PROP_OOP_INFOP property. See the *HP-UX 11i v3 Driver Development Guide* for more information.

**DL_HP_DRV_IPV6CKO_OUT**

For drivers that support outbound IPv6 CKO. The drivers setting this feature bit must get the inbound OOP template from HP-DLPI using the DLPI_PROP_OOP_INFOP property. This feature may be enabled/disabled after the initialization. Any change in feature must trigger an associated link DOWN and UP events. The change in this feature must also be followed by a DLPI_PROP_OOP_INFOP property. See the *HP-UX 11i v3 Driver Development Guide* for more information.

**DL_HP_DRV_COW**

Reserved for internal use.

**DL_HP_DRV_LFP**

Reserved for internal use.

**DL_HP_DRV_LNP**

Reserved for internal use.

**DL_HP_DRV_NO_SRC_ROUTING**

Reserved for internal use.
**DL_HP_DRV_NO_FASTPATH**
This flag must be set if the drivers does not support fast-path. See the *HP-UX 11i v3 Driver Development Guide* for more information.

**DL_HP_DRV_HP_APA**
Reserved for internal use.

**DL_HP_DRV_VLAN**
Driver supports VLANs.

**DL_HP_DRV_VTO**
Drivers support VLAN Tag Offload. Drivers insert VLAN tags (on outbound frames) and extract VLAN tags (on inbound frames).

**DL_HP_DRV_VI**
Reserved for internal use.

**DL_HP_DRV_PHYSICAL**
Set if the interface is a physical interface.

**DL_HP_DRV_LINKAGG**
Reserved for internal use.

**DL_HP_DRV_XPORT_OFFLOAD**
Reserved for internal use.

**DL_HP_DRV_LAN_CLASS**
Physical drivers register their class with the I/O subsystem. The instance numbers (passed in `dhc_instance_num` field of the `dl_hp_create_info_t` structure) are unique only within a driver class. All tightly coupled drivers must specify the `lan` class. Hence, all physical drivers that set `DL_HP_DRV_HP_DLS` must set this bit.

Loosely coupled drivers may or may not specify the `lan` class, during the registration with I/O subsystem. If they do belong to `lan` class, it implies that these driver (and their native DLPI) support LAN commands (`lanadmin` and `lanscan`) and their instance numbers are in the range allocated for `lan` class drivers. Such drivers must also set the `DL_HP_DRV_LAN_CLASS` bit in the features field during registration with the HP-DLPI. However, as these drivers are loosely coupled, the `DL_HP_DRV_HP_DLS` bit must not be set.

**DL_HP_DRV_NOMULTIFRAGCKO**
Driver does not support CKO for multi-fragmented packets.

Setting this bit without any of the `DL_HP_DRV_CKO*` features bit is invalid.

This flag must be set only if `DL_HP_DRV_TRAIN` is also set.

This feature may be enabled/disabled after the initialization. Any change in feature must trigger an associated link DOWN and UP events. See the *HP-UX 11i v3 Driver Development Guide* for more information.

**DL_HP_DRV_ILAN**
Reserved for internal use.

**DL_HP_DRV_TRAIN**
Driver supports packet trains.

**DL_HP_DRV_64BIT_MIB**
Drivers must set this bit during instance registration if the interface supports 64-bit MIB statistics. However, drivers can reset this feature bit anytime, if needed.

**DL_HP_DRV_MGMT_INSTANCE**
Reserved for internal use.

**DL_HP_DRV_PORT**
Drivers can set this bit to request the TCP/UDP layers to send down the source and destination TCP/UDP port numbers as part of the OOP, on a per-packet basis. See `dl_hp_oop_type_t` for more information.
Drivers that support TCP segmentation offload (TSO) for IPv4 packets must set this bit. The drivers setting this feature bit must get the inbound OOP template from HP-DLPI using the `DLPI_PROP_OOP_INFOP` property. This feature may be enabled and disabled after the initialization. Any change in feature must trigger an associated link DOWN and UP events. See the *HP-UX 11i v3 Driver Development Guide* for more information.

After setting this bit, the driver must call the `DL_HP_SPECIAL_PARAMS` property in order to set TCP segmentation related special parameters.

Any time you change this bit you must also call a `DLPI_PROP_OOP_INFOP` property.

Drivers that support TCP segmentation offload (TSO) for IPv6 packets must set this bit. The drivers setting this feature bit must get the inbound OOP template from HP-DLPI using the `DLPI_PROP_OOP_INFOP` property. This feature may be enabled and disabled after the initialization. Any change in feature must trigger an associated link DOWN and UP events. See the *HP-UX 11i v3 Driver Development Guide* for more information.

After setting this bit, the driver must call the `DL_HP_SPECIAL_PARAMS` property in order to set TCP segmentation related special parameters.

Any time you change this bit you must also call a `DLPI_PROP_OOP_INFOP` property.

Drivers, which have applications that directly interact with driver without going through DLPI, must set this feature bit so that the `DL_HP_USAGE_INFO_REQ` request will be forwarded to them and their specific usage information can be returned to upper layer together with those that are reported by DLPI. If this bit is not set, `DL_HP_USAGE_INFO_REQ` will not be forwarded to drivers. This features bit is only meaningful for a driver that uses HP-DLPI. If you implement your own DLPI, you might take advantage of this bit depending on your implementation.

Drivers that provided their own Critical Resource Analysis (CRA) subsystem will set this bit. This information will be passed to NetCRA through a `DL_HP_EXT_PPA_REQ` request, and NetCRA will not do CRA for those interface.

**NOTE:** A driver that sets this bit can still support the `DL_HP_USAGE_INFO_REQ` primitive, which can be used for other purposes.

**SEE ALSO**

`dlpi_propp(9F), dl_hp_create_info_t(9S), dl_hp_oop_type_t(9S)`
dl_hp_drv_param_req_type_t(9S)

NAME
dl_hp_drv_param_req_type_t - Sub-request type for DL_HP_SET_DRV_PARAM_IOCTL and
DL_HP_GET_DRV_PARAM_IOCTL.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
This structure provides an enumeration for the sub-request type that is used in the transparent
IOCTLs defined by HP-DLPI.

STRUCTURE MEMBERS
typedef enum dl_hp_drv_param_req_type {
    DL_HP_DRV_SPEED      = 0x1,
    DL_HP_DRV_DUPLEX     = 0x2,
    DL_HP_DRV_AUTONEG    = 0x4,
    DL_HP_DRV_MTU        = 0x8,
    DL_HP_DRV_RESET_MTU  = 0x10,
    DL_HP_VALUE1         = 0x20,
    DL_HP_VALUE2         = 0x40,
    DL_HP_VALUE3         = 0x80,
    DL_HP_RESERVED1      = 0x100,
    DL_HP_RESERVED2      = 0x200,
    DL_HP_RESERVED3      = 0x400,
    DL_HP_RESERVED4      = 0x800,
    DL_HP_SERIALIZE      = 0x40000000
} dl_hp_drv_param_req_type_t;

DL_HP_DRV_SPEED          Get or Set speed of the interface. The dl_speed field of
dl_hp_get_drv_param_ioctl_t or
dl_hp_set_drv_param_ioctl_t must be used to get or set
the driver speed.

DL_HP_DRV_DUPLEX         Get or Set Duplex mode of the interface. The dl_duplex field of
dl_hp_get_drv_param_ioctl_t or
dl_hp_set_drv_param_ioctl_t must be used to get or set
the driver duplex mode.

DL_HP_DRV_AUTONEG        Get or Set the Autonegotiation or Autosense value of the interface.
The dl_autoneg field of dl_hp_get_drv_param_ioctl_t or
dl_hp_set_drv_param_ioctl_t must be used to get or set
the driver's autonegotiation or autosense value.

DL_HP_DRV_MTU            Get or Set the MTU of the interface. The dl_mtu field of
dl_hp_get_drv_param_ioctl_t or
dl_hp_set_drv_param_ioctl_t must be used to get or set
the driver MTU.

DL_HP_DRV_RESET_MTU      Reset the MTU of the interface to its default value.

DL_HP_VALUE1             Driver-specific extension. The dl_value1 field of
dl_hp_get_drv_param_ioctl_t or
dl_hp_set_drv_param_ioctl_t may be used to get or set a
driver property.
Driver-specific extension. The dl_value2 field of dl_hp_get_drv_param_ioctl_t or dl_hp_set_drv_param_ioctl_t may be used to get or set a driver property.

Driver-specific extension. The dl_value3 field of dl_hp_get_drv_param_ioctl_t or dl_hp_set_drv_param_ioctl_t may be used to get or set a driver property.

Driver-specific extension. The dl_reserved1[0] field of dl_hp_get_drv_param_ioctl_t or dl_hp_set_drv_param_ioctl_t may be used to get or set a driver property.

Driver-specific extension. The dl_reserved1[1] field of dl_hp_get_drv_param_ioctl_t or dl_hp_set_drv_param_ioctl_t may be used to get or set a driver property.

Reserved for internal use by HP-DLPI.

Reserved for internal use by HP-DLPI.

Request HP-DLPI to serialize the request. Only applicable to DL_HP_SET_DRV_PARAM_IOCTL.

SEE ALSO

DL_HP_SET_DRV_PARAM_IOCTL(9G), DL_HP_GET_DRV_PARAM_IOCTL(9G)
NAME
dl_hp_duplex_mode_t - Duplex setting of the interface.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
Duplex setting of the interface. It is used in the dl_hp_op_param_t structure to set the duplex setting of the network interface in HP-DLPI.

STRUCTURE MEMBERS
typedef enum dl_hp_duplex_mode {
    DL_HP_HALF_DUPLEX,
    DL_HP_FULL_DUPLEX
} dl_hp_duplex_mode_t;

DL_HP_HALF_DUPLEX    Network interface is in half duplex mode.
DL_HP_FULL_DUPLEX    Network interface is in full duplex mode.

SEE ALSO
dlpi_propp(9F), dl_hp_op_param_t(9S)
dl.hp_encaps_type_t(9S)

NAME
dl.hp_encaps_type_t - Enumerate data type for packet encapsulation types.

SYNOPSIS
#include <sys/dlpi_ext.h>

DESCRIPTION
Encapsulation types supported by the interface

STRUCTURE MEMBERS
typedef enum {
    DL_HP_IEEE = 0x01,
    DL_HP_EXT_IEEE = 0x02,
    DL_HP_SNAP = 0x04,
    DL_HP_ETHERTYPE = 0x08,
    DL_HP_NOVELL = 0x10
} dl_hp_encaps_type_t;

DL_HP_IEEE IEEE 802.3 encapsulation.
DL_HP_EXT_IEEE IEEE 802.3 extended SAP encapsulation type.
DL_HP_SNAP SNAP 802.3 encapsulation type.
DL_HP_ETHERTYPE Ether encapsulation type.
DL_HP_NOVELL Novell encapsulation type.

SEE ALSO
dlpi_propp(9F)
NAME

dl_hp_event_link_cause_t - Cause for driver events.

SYNOPSIS

#include <sio/dlpi_drv.h>

DESCRIPTION

This structure provides an enumeration of hints sent using the `event_infop` parameter for
`DL_HP_EVENT_INTERFACE_UP` and `DL_HP_EVENT_INTERFACE_DOWN` events.

HP-DLPI does not in any way use the following hints, but may pass the information to other
interested modules. Hence, drivers are still expected to notify HP-DLPI of the events such as, a
change in IfOper/IfAdmin status, hardware state, link speed, MTU, MAC address, driver features
and so on, explicitly using the `dlpi_propp` property exchange mechanism.

STRUCTURE MEMBERS

typedef enum dl_hp_event_link_cause {
    DL_HP_EVENT_USER_RESET,
    DL_HP_EVENT_DRIVER_RESET,
    DL_HP_EVENT_OP_PARAM_CHANGE,
    DL_HP_EVENT_ADDR_CHANGE,
    DL_HP_EVENT_PCI_ERR,
    DL_HP_EVENT_PARENT_DOWN,
    DL_HP_EVENT_MEMBERS_DOWN,
    DL_HP_EVENT_CABLE,
    DL_HP_EVENT_HW_STATE_DOWN,
    DL_HP_EVENT_IFADMIN_DOWN,
    DL_HP_EVENT_FEATURES_CHANGE,
    DL_HP_EVENT_HA,
    DL_HP_EVENT_DEFAULT
} dl_hp_event_link_cause_t;

dl_hp_event_cause_t

DL_HP_EVENT_USER_RESET
    This hint is sent by the driver if the reset of the interface
    occurred due to the `DL_HP_HW_RESET` primitive.

DL_HP_EVENT_DRIVER_RESET
    This hint is sent by the driver if the reset is internally
    generated by the card/driver. This could occur due to a
    DMA timeout.

DL_HP_EVENT_OP_PARAM_CHANGE
    This hint is associated with change of an operational
    parameter for an interface.

DL_HP_EVENT_ADDR_CHANGE
    This hint is associated with a MAC address change.

DL_HP_EVENT_PCI_ERR
    The NIC detected a PCI error condition as the reason for
    the link going down.

DL_HP_EVENT_PARENT_DOWN
    Reserved for internal use.

DL_HP_EVENT_MEMBERS_DOWN
    Reserved for internal use.

DL_HP_EVENT_CABLE
    This hint is associated with a cable disconnect or reconnect.

DL_HP_EVENT_HW_STATE_DOWN
    This hint is associated with the NIC going to a dead state.

DL_HPEVENT_IFADMIN_DOWN
    This hint is associated with ifAdmin change. For a list of
    supported ifadmin values, see `dl_hp_ifadmin_state_t(9F).

DL_HP_EVENT_FEATURES_CHANGE
    This hint is associated with driver features change.
DL_HP_EVENT_HA
This hint is currently unused. It is reserved for future use.

DL_HP_EVENT_DEFAULT
This is the default event hint. Can be used for link UP events when driver does not know the exact cause.

SEE ALSO

dl_hp_ifadmin_state_(9S)
**NAME**

dl_hp_event_type_t - Driver events.

**SYNOPSIS**

`#include <sio/dlpi_drv.h>`

**DESCRIPTION**

This is a list of events that can be passed from the driver to the HP-DLPI event handler. The enumerated type defines the driver events that are supported by HP-DLPI. Each event is represented as a bit flag.

Some events have associated information passed through the `event_infop` parameter. For example, in the case of `DL_HP_EVENT_INTERFACE_DOWN` and `DL_HP_EVENT_INTERFACE_UP` events, the cause field is provided by the driver. The cause field is sent as the `event_infop` parameter of the event handler for link up or down events.

**STRUCTURE MEMBERS**

```c
typedef enum dl_hp_event_type {
    DL_HP_EVENT_INTERFACE_DOWN,
    DL_HP_EVENT_INTERFACE_UP,
    DL_HP_EVENT_OLD,
    DL_HP_EVENT_UNLOAD_CHECK,
    DL_HP_EVENT_CLEAR_STATS,
    DL_HP_EVENT_ARP_XMIT_REQ
} dl_hp_event_type_t;
```

**DL_HP_EVENT_INTERFACE_DOWN**

This event must be sent whenever the interface enters a state that disallows data transfer. That can occur due to any one of the following:

- Physical link went down.
- `ifAdmin` status changed to DOWN on the interface.
- Card being reset/suspended/dead.
- The driver detecting an error and thus transitioning to a state indicating error.
- Change in driver features.
- Change in MAC address and other operational parameters.

See `dl_hp_event_link_cause_t(9F)` for further information on filling the `event_infop` parameter.

**DL_HP_EVENT_INTERFACE_UP**

This event could occur due to any one of the following reasons:

- The physical link reconnected/recovered from an error state/resumed.
- The `ifAdmin` status changed to UP on the interface.
- The card reset is complete.
- Change in driver features.
- Change in MAC address and other operational parameters.
This event will be instrumental in informing the upper layers about change in the interface properties they have cached.

See `dl_hp_event_link_cause_t(9F)` for further information on filling the `event_infop` parameter.

### DL_HP_EVENT_OLD
This is not supported.

### DL_HP_EVENT_UNLOAD_CHECK
This is not supported.

### DL_HP_EVENT_CLEAR_STATS
This request must be issued by the driver after it clears its MIB statistics.

### DL_HP_EVENT_ARP_XMIT_REQ
This event is reserved for internal use only.

**SEE ALSO**

`dl_hp_event_link_cause_t(9S)`
NAME

DL_HP_GET_DRV_PARAM_IOCTL - HP-DLPI defined transparent IOCTL to get driver parameters.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

This IOCTL is sent to drivers using the driver_controlp routine. The associated structure for this request is dl_hp_get_drv_param_ioctl_t structure.

This IOCTL is defined by HP-DLPI for usage by drivers. This IOCTL is transparent to HP-DLPI. Transparent IOCTLs are the ones that are not interpreted by HP-DLPI and are sent directly to the driver.

This IOCTL can be used to get driver operating parameters like Speed, Duplicity, MTU and Autonegotiation. Drivers can add new sub-requests to satisfy any driver specific commands.

STRUCTURE MEMBERS

typedef struct dl_hp_get_drv_param_ioctl {
    dl_hp_drv_param_req_type_t dl_request;
    uint32_t dl_speed;
    dl_hp_duplex_mode_t dl_duplex;
    dl_hp_autoneg_sense_t dl_autoneg;
    uint32_t dl_mtu;
    uint64_t dl_value1;
    int64_t dl_value2;
    uint64_t dl_value3;
    uint32_t dl_reserved1[2];
    uint32_t dl_reserved2[2];
} dl_hp_get_drv_param_ioctl_t;

dl_request The sub-request. The driver properties that need to be retrieved (see dl_hp_drv_param_req_type_t(9F)). The caller can set multiple requests in dl_request. Depending on the requests set in dl_request, the driver may return values in one or more of the following fields.

dl_speed Speed of underlying link in Mbits/sec.

dl_duplex Duplex modes. The field is of type dl_hp_duplex_mode_t.

dl_autoneg Auto-negotiation/auto-sense. The field is of type dl_hp_autoneg_sense_t.

dl_mtu MTU of the interface in octets.

dl_value1 64 bit value for driver extension.

dl_value2 64 bit value for driver extension.

dl_value3 64 bit value for driver extension.


dl_reserved2[2] Reserved for internal use by HP-DLPI. Must be set to 0.

CONSTRAINTS

HP-DLPI does not interpret this request. This request must be used between the driver’s user space application and the driver.
The DL_HP_SERIALIZE bit in dl_hp_drv_param_request_type_t is ignored for DL_HP_GET_DRV_PARAM_IOCTL.

The drivers can define their own dl_request by using any other bits that are not defined by HP-DLPI.

It is driver's responsibility to check the privileges of the user who issued the request (using ioc_cr field of iocblk).

The driver must fill up the ioc_count value in the IOCTL MBLK if the request is successfully processed.

SEE ALSO
dlpicontrolp(9F), dl_hp_drv_param_req_type_t(9S), dl_hp_duplex_mode_t(9S), dl_hp_autoneg_sense_t(9S)
NAME

DL_HP_GET_MIBSTATS_REQ - Request to get the physical interface statistics.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

This primitive is sent to drivers using the driver_controlp routine to get the physical interface statistics. The associated structure for this request is dl_hp_get_mibstats Req_t structure. This primitive only applicable between HP-DLPI and the driver.

STRUCTURE MEMBERS

typedef struct {
    uint32_t dl_primitive;
    uint32_t dl_reserved1;
    uint32_t dl_reserved2;
} dl_hp_get_mibstats Req_t;

dl_primitive Set to DL_HP_GET_MIBSTATS_REQ
dl_reserved1 Reserved for future use. Set to 0.
dl_reserved2 Reserved for future use. Set to 0.

CONSTRAINTS

The second MBLK (linked by b_cont) will contain space to hold the statistics of size Ext_mib_t. The second MBLK will be empty (b_rptr == b_wptr). After writing the information (statistics information) to the second MBLK, the drivers must move the b_wptr to the size that was written to the second MBLK (in this case by the size of the Ext_mib_t). The drivers can cast the b_rptr to be Ext_mib_t *, if it has to change some fields.

Drivers must not modify the b_rptr.

Even if driver supports only a subset of the statistics groups in the Ext_mib_t structure, it must copy the complete structure.

Ext_mib_t is defined in <sys/mib.h>

SEE ALSO

driver_controlp(9F)
NAME

DL_HP_GET_64BIT_STATS_REQ - Request to get 64-bit statistics from an interface.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

This primitive is sent to drivers using the driver_controlp routine to get the 64-bit statistics. The associated structure for this request is dl_hp_get_64bit_stats_req_t.

STRUCTURE MEMBERS

typedef struct {
    uint32_t        dl_primitive;
    uint32_t        dl_reserved1;
    uint32_t        dl_reserved2;
} dl_hp_get_64bit_stats_req_t;

dl_primitive Set to DL_HP_GET_64BIT_STATS_REQ

dl_reserved1 Reserved for future use. Set to 0.

dl_reserved2 Reserved for future use. Set to 0.

CONSTRAINTS

The second MBLK (linked by b_cont) will contain the space to hold the statistics of size Ext_64bit_mib_t.

Ext_64bit_mib_t is defined in <sys/mib.h>.

The second MBLK will be empty (b_rptr == b_wptr). After writing the information (include both 64-bit and standard 32-bit statistics) to the second MBLK, the drivers must move the b_wptr to the size that was written to the second MBLK. (In this case, by the size of the Ext_64bit_mib_t).

Drivers must not modify the b_rptr.

If the driver does not have any interface specific statistics, the b_wptr must still be set to the end of Ext_64bit_mib_t, and zero the interface-specific part of the data structure.

The driver is responsible for following RFC 2863 and other RFCs, which define the interface specific MIBs, to fill out all the fields.

For drivers that support 64-bit statistics, all 32-bit statistics requests will be converted to DL_HP_GET_64BIT_STATS_REQ if the feature bit DL_HP_DRV_64BIT_MIB is set when the request is received. If a driver does not change this bit, no 32-bit request will be received. However, if a driver dynamically resets this feature bit, it will receive both 32-bit and 64-bit statistics requests depending on whether the features bit is set at the point the request is received by DLPI.

SEE ALSO

driver_controlp(9F)
NAME

dl_hp_getinfo_t - Generic data type to get information from HP-DLPI.

SYNOPSIS

#include <sio/dlpi_drv.h>

DESCRIPTION

The data type that must be passed in the `valuep` parameter to `dlpi_propp` for the
`DLPI_PROP_MCAST_INFOP` and `DLPI_PROP_INSTANCE_INFOP` properties.

Except for the `dhg_modifier` field, all other fields are set by `dlpi_propp` on a successful
return.

The data type for each item in the definition depends on the property being requested. The
`dhg_datap` being returned could be an MBLK or a memory block (see the property definitions
in `dl_hp_prop_t(9F)` for the type of `dhg_datap` returned). The caller must not modify the contents
of the MBLK or data block in the `dhg_datap` in any way and must invoke the function provided
(`dhg_free_funcp`).

The caller must always use `dhg_len` to walk through the list of items in the buffer (the caller
must not use `sizeof` to determine the size of each item.)

STRUCTURE MEMBERS

typedef struct dl_hp_getinfo {
    dl_hp_search_info_t  *dhg_modifier;
    uint32_t              dhg_count;
    uint32_t              dhg_len;
    void                 (*dhg_free_funcp)(uint8_t *datap);
    uint8_t              *dhg_datap;
    void                 *dhg_rsvdp;
} dl_hp_getinfo_t;

dhg_modifier   Caller must set this to NULL unless the property allows specification of
                some criteria to limit the information requested. For example, the
                `DLPI_PROP_INSTANCE_INFOP` property allows the caller to request
                information about driver instances that match a specific criteria. In this
                case, this field must point to structure (for example,
                `dl_hp_search_info_t`) specifying the search criteria.
                When calling `DLPI_PROP_MCAST_INFOP`, this element must be set to
                NULL.

    dhg_count    HP-DLPI will set this field. Specifies the total number of items being
                returned.

    dhg_len      HP-DLPI will set this field. Specifies the length of each item.

    dhg_free_funcp   HP-DLPI will set this field.

                Specifies the callback function that must be called after the driver is done
                processing the requested information passed in the `dhg_datap` field.
                This function must be passed the pointer passed in `dhg_datap`.
                The caller must not modify `dhg_datap`.

    dhg_datap    If `dhg_count` is set to 0, HP-DLPI was unable to find a match based on
                the search criteria specified by the caller. In which case, `dhg_datap` will
                be set to NULL. If HP-DLPI is able to find a match, it contains a pointer
to a memory block of size \texttt{dhg\_len} times \texttt{dhg\_count}. This is the information returned to the caller.

Set to NULL if no information was returned (that is, \texttt{dhg\_count} was 0). If information was returned, each item is of length \texttt{dhg\_len}. The \texttt{dhg\_len} must be used to walk through the list of items in the data passed.

The data type of the item returned depends on the property requested and is described in the definition of the property. After reading the data passed, the caller must call the \texttt{dhg\_free\_funcp} function pointer with \texttt{dhg\_datap} as the parameter. See \texttt{dl\_hp\_prop\_t(9F)} for more information.

\texttt{dhg\_rsvdp}  
Reserved for future use. Must be set to 0.

**SEE ALSO**

\texttt{dlpi\_propp(9F)}, \texttt{dl\_hp\_prop\_t(9S)}, \texttt{dl\_hp\_search\_info\_t(9S)}, \texttt{sizeof(9F)}
dl_hp_get_tcpseg_info_t(9S)

**NAME**
dl_hp_get_tcpseg_info_t - Structure containing TCP segmentation offload parameters supported by drivers.

**SYNOPSIS**
#include <sio/dlpi_drv.h>

**DESCRIPTION**
Specification to get or set TCP segmentation offload parameters. Used when invoking the DLPI_PROP_SPECIAL_PARAMS properties.

**STRUCTURE MEMBERS**

typedef struct dl_hp_get_tcpseg_info {
    uint32_t             dl_tcpss_version;
    uint32_t             dl_tcpss_vmtu;
    uint32_t             dl_tcpss_min;
    dl_hp_tcpseg_flags_t dl_tcpss_flags;
} dl_hp_get_tcpseg_info_t;

dl_tcpss_version Specifies the structure version number.
dl_tcpss_vmtu Virtual MTU of the interface. This is the maximum payload size that is supported by the NIC. Datagrams less than or equal to this size are offloaded to the NIC for segmentation.
dl_tcpss_min Minimum number of new segments that the NIC must be capable of deriving from the original segment. For example, if this is set to N, the stack must not send the segment to the NIC unless it can be broken into more than N segments by the NIC.
dl_tcpss_flags One or more of the following flags:

**IP_OPTIONS**
Set by the driver to indicate that the NIC supports IP options. If set, all IP options are supported and copied into each of the newly segmented frames. If any of the options are required to be modified or need to be different from the other frames, that segment must be sent to the NIC for segmentation offload.

**TCP_OPTIONS**
Set by the driver to indicate that the NIC supports TCP options. If set, all TCP options are supported and copied into each of the newly segmented frames. If any of the options are required to be modified or need to be different from the other frames, that segment must be sent to the NIC for segmentation offload.

**HEADER_BUFFER_ALLOC**
Indicates that the NIC has some restrictions in the way the data is divided among the various buffers. If set, the stack must send all of the headers (link, IP and TCP header) in a single mblk.
SEE ALSO

dl_hp_sp_param_type(9S), dl_hp_sp_params_t(9S), dl_hp_tcbseg_info_t(9S)
NAME

dl_hp_hdr_length_t - Data type for OOP type DL_HP_OOP_HDR_LENGTH.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

Contains the length of the LLC header of the packet. This OOP type is defined in
dl_hp_oop_type_t.

STRUCTURE MEMBERS

typedef struct dl_hp_hdr_length {
    uint16_t       dhhl_reserved;
    uint16_t       dhhl_length;
} dl_hp_hdr_length_t;

dhhl_reserved    Reserved element.
dhhl_length      LLC plus OOP header length.

SEE ALSO

dl_hp_oop_type_t(9S)
NAME

DL_HP_HW_RESET_REQ - Request to reset physical interface.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

This primitive is sent to drivers using the driver_controlp routine to reset the physical interface. The associated structure for this request is dl_hp_hw_reset_req_t structure.

The DL_HP_HW_RESET_REQ primitive described in this manpage may be different from the definition in DLPI Programmer’s Guide. The information provided here is applicable only to tightly-coupled drivers.

STRUCTURE MEMBERS

typedef struct {
    uint32_t dl_primitive;
    dl_hp_hw_reset_req_t;
} dl_primitive

dl_primitive Set to DL_HP_HW_RESET_REQ

CONSTRAINTS

Drivers need to generate link events (link DOWN and UP) when there is a change in any of the driver properties (in this case the operating state may change). See the HP-UX 11i v3 Driver Development Guide for information on event generation.

SEE ALSO

driver_controlp(9F)
NAME
dl_hp_hw_state_t - Hardware state of the interface.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
Hardware state of the interface. This enumeration is used to set hardware state using the
DLPI_PROP_HDW_STATE property.

STRUCTURE MEMBERS
typedef enum dl_hp_hw_state {
    DL_HP_HW_UP,
    DL_HP_HW_DOWN
} dl_hp_hw_state_t;

DL_HP_HW_UP Hardware state is UP.
DL_HP_HW_DOWN Hardware state is DOWN.

SEE ALSO
dlpi_propp(9F)
NAME
dl_hp_ifadmin_state_t - Administration state of the interface (ifAdminStatus).

SYNOPSIS
#include <sys/dlpi_ext.h>

DESCRIPTION
The desired state of the interface. This definition is used to set ifAdmin state of the interface using the DLPI_PROP_IFADMIN property.

STRUCTURE MEMBERS
typedef enum dl_hp_ifadmin_state {
    DL_HP_IFADMIN_UP,
    DL_HP_IFADMIN_DOWN,
    DL_HP_IFADMIN_TEST
} dl_hp_ifadmin_state_t;

DL_HP_IFADMIN_UP Administration status of the interface is UP.
DL_HP_IFADMIN_DOWN Administration status of the interface is DOWN.
DL_HP_IFADMIN_TEST Not Supported.

SEE ALSO
dlpi_propp(9F)
dl_hp_info_t(9S)

NAME
dl_hp_info_t - Union of all types of usage information data structures.

SYNOPSIS
#include <sio/dlpi_ext.h>

DESCRIPTION
This data type carries one usage information entry. Depending on the usage type, the dl_info field must be interpreted using different structure.

STRUCTURE MEMBERS
typedef union dl_hp_info {
    dl_hp_cmd_info_t dl_cmd_info
    dl_hp_vlan_info_t dl_vlan_info;
    dl_hp_apa_info_t dl_apa_info;
    dl_hp_toe_info_t dl_toe_info;
    uint32_t dl_reserved[2];
} dl_hp_info_t;

dl_cmd_info Element of type dl_hp_cmd_info_t contains an application that is attached to the PPA. This information is present if the DL_HP_CMD_INFO type is set in the dl_info_type field of the dl_hp_usage_info_t structure.

dl_vlan_info Element of type dl_hp_vlan_info_t contains an VLAN's ppa and vlanid, which is created on the interface. Present if the DL_HP_VLAN_INFO type is set in the dl_info_type field of the dl_hp_usage_info_t structure.

dl_apa_info Reserved for internal use.

dl_toe_info Reserved for internal use.

dl_reserved Used to ensure that the size of the data structure is at least 8 bytes.

SEE ALSO
dl_hp_cmd_info_t(9S), dl_hp_info_type_t(9S), dl_hp_usage_info_t(9S), dl_hp_vlan_info_t(9S)
dl_hp_info_type_t(9S)

NAME
dl_hp_info_type_t - Enumerated usage information type contained in dl_hp_info_t.

SYNOPSIS
#include <sio/dlpi_ext.h>

DESCRIPTION
This structure defines the enumeration for usage information types that are used to indicate the current type of dl_info in the dl_hp_usage_info_t data structure.

STRUCTURE MEMBERS
typedef enum dl_hp_info_type {
     DL_HP_CMD_INFO = 1,
     DL_HP_VLAN_INFO,
     DL_HP_APA_INFO,
     DL_HP_TOE_INFO
} dl_hp_info_type_t;

DL_HP_CMD_INFO Indicates the dl_info element is to be interpreted as dl_hp_cmd_info_t, which is used to report applications that are attached to PPA.

DL_HP_VLAN_INFO Indicates the dl_info field is to be interpreted as a dl_hp_vlan_info_t type, which is used to report an VLAN interface created on the underlying physical or link aggregation interface.

DL_HP_APA_INFO Reserved for internal use.

DL_HP_TOE_INFO Reserved for internal use.

SEE ALSO
dl_hp_usage_info_t(9S), dl_hp_info_t(9S), dl_hp_cmd_info_t(9S), dl_hp_vlan_info_t(9S)
NAME

dl_hp_instance_info_t - Structure returned on use of DL_PROP_INTERFACE_INFOP HP-DLPI property.

SYNOPSIS

#include <sio/dlpi_drv.h>

DESCRIPTION

This structure is returned in response to the use of the DL_PROP_INTERFACE_INFOP HP-DLPI property. This structure contains the interface information returned by a DLS provider.

STRUCTURE MEMBERS

typedef struct dl_hp_instance_info {
    uint32_t dii_version;
    uint32_t dii_instance_num;
    dl_hp_mac_type_t dii_mac_type;
    uint32_t dii_mac_addr_len;
    uint32_t dii_nm_id;
    dl_hp_encaps_type_t dii_encaps;
    uint32_t dii_mtu;
    uint32_t dii_major_num;
    dl_hp_drv_features_one_t dii_features_one;
    uint32_t dii_features_two;
    uint32_t dii_features_three;
    uint8_t dii_mac_addr[DL_HP_MAX_MAC_ADDR_LEN];
    char dii_hdw_path[DL_HP_MAX_HDW_PATH_LEN];
    char dii_drv_name[DL_HP_MAX_DRIVER_NAME_LEN];
    char dii_arpmod_name[DL_HP_MAX_ARPMOD_NAME_LEN];
    uint32_t dii_max_mcast;
    uint32_t dii_max_vlan;
    dl_hp_ifadmin_state_t dii_ifadmin;
    dl_hp_link_state_t dii_ifoper;
    dl_hp_duplex_mode_t dii_duplex;
    uint32_t dii_speed;
    uint32_t (*dii_outputp)(void *drv_output_hdlp,
                            mblk_t *mblkp,
                            dl_hp_pkt_type_t pkt_type,
                            void *rsvd1p);
    uint32_t (*dii_build_hdrp)(void *drv_output_hdlp,
                                dl_hp_src_llc_info_t *src_infop,
                                dl_hp_dest_llc_info_t *dset_infop,
                                dl_hp_llc_info_t *hdr_infop,
                                void *rsvd1lp,
                                void *rsvd2p);
} dl_hp_instance_info_t;
void *dii_output_hdlp;
void *dii_dlpi_hdlp;
uint32_t dii_ppa;
uint32_t dii_next_offset;
uint64_t dii_reserved[2];
} dl_hp_instance_info_t;

dii_version The lowest HP-DLPI version with which this interface is compatible.
dii_instance_num Maps one-to-one with a physical interface belonging to a particular class.
dii_mac_type This is a MAC type from one of the supported MAC types in the dl_hp_mac_type_t structure.
dii_mac_addr_len The length, in bytes, of the MAC address.
dii_nm_id The Network Management ID. Each interface has a unique NMID.
dii_encaps Specifies the encapsulations supported by the drivers. For supported encapsulations, see dl_hp_encaps_type_t(9S).
dii_mtu The current MTU of the link in bytes.
dii_major_num Identifies DLPI's major number. For loosely-coupled drivers, the major number is different from HP-DLPI's major number.
dii_features_one Identifies the list of supported features by the driver instance. For a list of supported features, see dl_hp_drv_features_one_t(9S).
dii_features_two This field is reserved for future use. It must be set to 0.
dii_features_three Reserved for future use. It must be set to 0.
dii_mac_addr The MAC address of the card. The size of MAC address must be equal to dii_mac_addr_len.

The MAC address is in canonical format for Ethernet and FDDI and in wire format for Token Ring.
dii_hdw_path The hardware path of the physical interface as a (null terminated) character string. The length of the hardware path could be a maximum of DL_HP_MAX_HDW_PATH_LEN bytes, including NULL termination. DL_HP_MAX_HDW_PATH_LEN is currently defined as 100.
dii_drv_name The driver name as a (null terminated) character string. The name must be less than or equal to DL_HP_MAX_DRIVER_NAME_LEN characters and must be NULL terminated. DL_HP_MAX_DRIVER_NAME_LEN is currently defined as 64.
dii_arpmod_name A character string that identifies the ARP (STREAMS) module name for this driver, if one is present. This field is currently used by Token Ring drivers.

The name must be less than or equal to DL_HP_MAX_ARPMOD_NAME_LEN characters and must be NULL terminated. DL_HP_MAX_ARPMOD_NAME_LEN is currently defined as 64.
dii_max_mcast The driver specified maximum number of unique multicast addresses supported by the driver or NIC in the range 1-0x7FFFFFFF.
dii_max_vlan Not supported in HP-UX 11i v3.
dii_ifadmin Identifies the administration status (ifAdmin) of the interface. For a list of supported ifadmin values, see dl_hp_ifadmin_state_t(9S).
**dii_ifoper**
Identifies the operational status (**ifOper**) of the interface. For a list of supported **ifOper** values, see *dl_hp_link_state_t*(9S).

**dii_duplex**
Specifies the duplicity setting of the driver instance. For supported values of duplicity, see *dl_hp_duplex_t*(9S).

**dii_speed**
Specifies the speed of the interface in Mb/s.

**dii_outputp**
The driver outbound routine. See *driver_outputp*(9F) for information.

**dii_build_hdrp**
The function that identifies the driver routine to build the MAC/LLC header in the outbound path. See *driver_build_hdrp*(9F) for information.

**dii_output_hdlp**
The driver-specified opaque (to HP-DLPI) handle that is passed to the **dii_outputp** and **dii_build_hdrp** outbound routines.

**dii_dlpi_hdlp**
HP-DLPI's per-interface handle. This is the opaque handle that drivers must use in calls to HP-DLPI interfaces.

**dii_ppa**
Specifies the physical point of attachment (PPA) of the interface.

**dii_next_offset**
Specifies the offset from the beginning of the returned memory block, to the next *dl_hp_instance_info_t* structure.

**dii_reserved**
Reserved for future use. It must be set to 0.

**SEE ALSO**
*dlpi_propp*(9F), *dl_hp_mac_type_t*(9S), *dl_hp_encaps_type_t*(9S), *dl_hp_features_one_t*(9S), *dl_hp_ifadmin_state_t*(9S), *dl_hp_link_state_t*(9S), *dl_hp_duplex_t*(9S), *driver_outputp*(9F), *driver_build_hdrp*(9F), *dl_hp_instance_info_t*(9S)
dl_hp_llc_info_t(9S)

NAME
dl_hp_llc_info_t - LLC header information.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
This structure contains the LLC header information. This structure is returned from driver's build header routine.

STRUCTURE MEMBERS
typedef struct dl_hp_llc_info {
    dl_hp_pkt_type_t          dli_pkt_type;
    uint32_t                  dli_llc_length;
    mblk_t                   *dli_mblkp;
    uint64_t                  dli_reserved1[2];
    uint64_t                  dli_reserved2[2];
} dl_hp_llc_info_t;

dli_pkt_type          Identifies the packet type. See dl_hp_pkt_type_t(9S) for a list of supported packet types.
dli_llc_length       Identifies the length of LLC header, in bytes. This does not include the size of OOP data, if any.
dli_mblkp            A pointer to a MBLK that contains LLC header and space for OOP data, if any.
dli_reserved1        Reserved for future use. It must be set to 0.
dli_reserved2        Reserved for future use. It must be set to 0.

SEE ALSO
dl_hp_pkt_type_t(9S)
NAME
dl_hp_mac_type_t -- MAC Type definitions.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
MAC Type of the interface. The dhc_mac_type element of dl_hp_create_info_t is of this type. The dl_hp_create_info_t structure is the associated structure for the DL_HP_OP_CREATE operation in dlpi_propp.

STRUCTURE MEMBERS

typedef enum dl_hp_mac_type {
    DL_HP_DEV_8023,
    DL_HP_DEV_8024,
    DL_HP_DEV_8026,
    DL_HP_DEVEther,
    DL_HP_DEV_HDLC,
    DL_HP_DEV_CHAR,
    DL_HP_DEV_CTCA,
    DL_HP_DEV_OTHER,
    DL_HP_DEV_FC,
    DL_HP_DEV_ATM,
    DL_HP_DEV_IPATM,
    DL_HP_DEV_X25,
    DL_HP_DEV_ISDN,
    DL_HP_DEV_HIPPI,
    DL_HP_DEV_100VG,
    DL_HP_DEV_100VGTPR,
    DL_HP_DEV_ETH_CSMA,
    DL_HP_DEV_100BT,
    DL_HP_DEV_IB
} dl_hp_mac_type_t;

HP-DLPI supports the following MAC types for tightly-coupled drivers:
DL_HP_DEV_8023 — IEEE 8023
DL_HP_DEV_Ether — Ethernet Interface (802.3 encapsulation is assumed to be supported if the driver sets its MAC type to this value).

Loosely-coupled drivers can be any of the types defined in the structure.

SEE ALSO
dl_hp_create_info_t(9S), dlpi_propp(9F)
dl_hp_oop_hdr_t(9S)

NAME
dl_hp_oop_hdr_t - Structure defining each OOP's header.

SYNOPSIS
#include <sys/dlpi_ext.h>

DESCRIPTION
OOP header. Structure definition for the OOP type header containing the type and size values.

STRUCTURE MEMBERS

typedef struct dl_hp_oop_hdr {
    uint16_t dhoh_type;
    uint16_t dhoh_size;
} dl_hp_oop_hdr_t;

dhoh_type OOP type. See dl_hp_oop_type_t(9S) for more information.
dhoh_size Size of the OOP data.

SEE ALSO

dl_hp_oop_type_t(9S)
dl_hp_oop_out_template_t(9S)

NAME

dl_hp_oop_out_template_t - Outbound OOP template.

SYNOPSIS

#include <sio/dlpi_drv.h>

DESCRIPTION

HP-DLPI uses the information in this structure to construct the outbound OOP template when requested by a HP-DLPI user. See the Driver Development Guide for more information.

STRUCTURE MEMBERS

typedef struct dl_hp_oop_out_template {
    uint32_t  doot_oop_size;
    mblk_t   *doot_mblkp;
    void     (*doot_free_funcp)(mblk_t *);
    void     *rsvdp;
    uint32_t  reserved[2];
} dl_hp_out_template_t;

doot_oop_size  Identifies the driver’s outbound OOP data size in bytes.
doot_mblkp     Pointer to the MBLK that contains the outbound OOP data template.
doot_free_funcp Identifies the function to free the doot_mblkp.
rsvdp          Reserved for future use. It must be set to NULL.
reserved       Reserved for future use. It must be set to zero (0).

SEE ALSO

dlpi_propp(9F)
**NAME**

dl_hp_oop_template_t- Structure to be passed when using the DLPI_PROP_OOP_INFOP property.

**SYNOPSIS**

```
#include <sio/dlpi_drv.h>
```

**DESCRIPTION**

The structure containing inbound OOP data template and the associated information. The OOP template passed in this structure must be used by the driver for every packet that is passed upstream to HP-DLPI.

The `dhot_mblkp` returned by HP-DLPI must never be freed. The `dhot_mblkp` present in subsequent calls to DLPI_PROP_OOP_INFOP will overwrite the previously returned `dhot_mblkp`.

For more information on OOP, see the HP-UX 11i v3 Driver Development Guide.

**STRUCTURE MEMBERS**

```c
typedef struct dl_hp_oop_template {
  mblk_t         *dhot_mblkp;
  uint32_t        dhot_drv_pad;
  uint32_t        dhot_oop_size;
  uint32_t        dhot_oop_copy_size;
  void           *rsvdp;
  uint32_t        reserved[2];
} dl_hp_oop_template_t;
```

- **dhot_mblkp**
  HP-DLPI sets `dhot_mblkp` pointer to the MBLK that contains the OOP data template. Valid only if `dhot_oop_size` is valid. Drivers must not free this MBLK at any time.

- **dhot_drv_pad**
  Drivers will set this element to specify the padding in the inbound packet required to avoid IP pull-ups. See HP-UX 11i v3 Driver Development Guide for more information.

- **dhot_oop_size**
  HP-DLPI will pass OOP data size in bytes.

- **dhot_oop_copy_size**
  The end-of-options OOP may contain padding and other non-data bytes. Because the drivers do not need to copy them and because they might make `dhot_oop_size` a non-multiple of 8 bytes, the copyable portion of the OOP template is specified in `dhot_oop_copy_size`. HP-DLPI guarantees that the copyable size will always be a multiple of 8.

- **rsvdp**
  Reserved for future use. Must be set to NULL.

- **reserved**
  Reserved for future use. Must be set to 0.

**SEE ALSO**

`dlpi_propp(9F)`
NAME
dl_hp_oop_type_t - Out-of-Packet data types.

SYNOPSIS
#include <sys/dlpi_ext.h>

DESCRIPTION
These enumerated types are OOP data types that are passed between HP-DLPI and the driver. In the inbound and outbound path, the actual payload of the packet is preceded by the OOP data based on the feature bits set by the driver in dhc_features_one element of dl_hp_create_info_t.

The OOP data communicated between HP-DLPI and the driver is different from the OOP data communicated between HP-DLPI and the upper layer. See the HP-UX 11i v3 Driver Development Guide for more information OOP mechanism between HP-DLPI and the driver.

Each OOP is a combination of Type-Size-Data. Each data type has a structure associated with it. OOP itself has a header of its own containing the Type-Size of OOP data. The structure is dl_hp_oop_hdr_t. See each OOP type for associated structures.

STRUCTURE MEMBERS
typedef enum dl_hp_oop_type {  
    DL_HP_OOP_PAD = 0x0001,  
    DL_HP_OOP_CKO = 0x0002,  
    DL_HP_OOP_VLAN_TAG = 0x0003,  
    DL_HP_OOP_TOS = 0x0004,  
    DL_HP_OOP_HDR_LENGTH = 0x0005,  
    DL_HP_OOP_TCPSEG_IPV4 = 0x0006,  
    DL_HP_OOP_TCPSEG_IPV6 = 0x0007,  
    DL_HP_OOP_PORT = 0x0008,  
    DL_HP_OOP_EOO = 0x00ff,  
    DL_HP_TYPE_DATA_INVALID = 0x1000,  
    DL_HP_TYPE_MASK = 0xefff  
} dl_hp_oop_type_t;

DL_HP_OOP_PAD          Padding type.
DL_HP_OOP_CKO          Reserved for future use.
DL_HP_OOP_VLAN_TAG     Reserved for future use.
DL_HP_OOP_TOS          Reserved for internal use.
DL_HP_OOP_HDR_LENGTH   Length of the header (OOP + LLC header) in a packet. Associated information for this OOP type is dl_hp_hdr_length_t.

DL_HP_OOP_TCPSEG_IPV4  Reserved for future use. Do not set this bit.
DL_HP_OOP_TCPSEG_IPV6  Reserved for future use. Do not set this bit.
DL_HP_OOP_PORT         Reserved for future use. Do not set this bit.
DL_HP_OOP_EOO          End of Options. Delimiter which separates the OOP and the payload.
DL_HP_TYPE_DATA_INVALID If this bit is set in any of the options types, the data associated with the type is be invalid.
DL_HP_TYPE_MASK        Mask to get the actual OOP type values.
SEE ALSO

dl_hp_hdr_length_t(9S), dl_hp_oop_hdr_t(9S), dl_hp_pri_info_t(9S), dl_hp_vlan_tag_t(9S)
**NAME**

dl_hp_op_param_t - HP-DLPI property to set driver operating parameters using DLPI_PROP_OP_PARAM property.

**SYNOPSIS**

```
#include <sio/dlpi_drv.h>
```

**DESCRIPTION**

This structure is used to set one or more operating parameters of a driver instance. To change an operating parameter, this structure is used with DLPI_PROP_OP_PARAM property. This property is valid for both tightly coupled and loosely coupled drivers.

**STRUCTURE MEMBERS**

```c
typedef struct dl_hp_op_param {
    dl_hp_op_param_type_t dop_param_type;
    uint32_t dop_speed;
    dl_hp_duplex_mode_t dop_duplex;
    uint32_t dop_mtu;
    dl_hp_autoneg_sense_t dop_auto_neg_sense;
    uint32_t dop_reserved[2];
} dl_hp_op_param_t;
```

- **dop_param_type**
  The HP-DLPI user will set this field. This indicates the one or more (up to a maximum of four) operating parameters that the DLS user can set at a time. For supported changes to operating parameters, see `dl_hp_op_param_type_t(9F)`.

- **dop_speed**
  The HP-DLPI user will set this field. This field specifies the speed of the driver instance in Mbits/sec.

- **dop_duplex**
  The HP-DLPI user will set this field. This field specifies the duplicity of the driver instance (full-duplex or half-duplex). For supported values of duplicity, see `dl_hp_duplex_mode_t(9F)`.

- **dop_mtu**
  The HP-DLPI user will set this field. This field specifies the MTU size in bytes for the driver instance. The minimum valid MTU is 10 and the minimum recommended value is 256.

- **dop_auto_neg_sense**
  The HP-DLPI user will set this field. This field specifies the autonegotiation value for Ethernet and FDDI drivers and autosense value for Token Ring drivers. For supported values of autonegotiation/autosense, see `dl_hp_autoneg_sense_t(9F)`.

- **dop_reserved[2]**
  This field is reserved for future use. It must be set to 0.

**SEE ALSO**

`dlpi_propp(9F), dl_hp_autoneg_sense_t(9S), dl_hp_duplex_mode_t(9S), dl_hp_op_param_type_t(9S)`
dl_hp_op_param_type_t(9S)

NAME
dl_hp_op_param_type_t - Operating parameters that drivers need to set.

SYNOPSIS
#include <sys/dlpi_ext.h>

DESCRIPTION
The enumerated type is used to specify which of the driver’s operating parameters are being set. These are used in the dl_hp_op_param_t structure. This enumeration contains bit wise flags to allow setting of multiple operating parameters for a driver instance.

STRUCTURE MEMBERS

typedef enum dl_hp_op_param_type {
    DL_HP_SPEED_SET,
    DL_HP_DUPLEX_SET,
    DL_HP_MTU_SET,
    DL_HP_AUTO_NEG_SENSE_SET
} dl_hp_op_param_type_t;

DL_HP_SPEED_SET Sets the speed of the interface. The dop_speed element of the dl_hp_op_param_t structure contains the valid speed value.

DL_HP_MTU_SET Sets the MTU of the interface. The dop_mtu element of the dl_hp_op_param_t structure contains the valid MTU value.

DL_HP_DUPLEX_SET Sets the duplex type of the interface. The dop_duplex element of the dl_hp_op_param_t structure contains the valid duplex setting.

DL_HP_AUTO_NEG_SENSE_SET Sets the Auto Negotiation/Auto Sense value of the interface. The dop_auto_neg_sense element of the dl_hp_op_param_t structure contains valid Auto Negotiation/Auto Sense value.

SEE ALSO
dlpi_propp(9F), dl_hp_op_param_t(9S)
dl_hp_op_t(9S)

NAME

dl_hp_op_t - Operation codes supported for dlpi_propp routine.

SYNOPSIS

#include <sio/dlpi_drv.h>

DESCRIPTION

Operators for the HP-DLPI dlpi_propp property exchange interface. This corresponds to the opcode argument in dlpi_propp.

STRUCTURE MEMBERS

typedef enum dl_hp_op {
    DL_HP_OP_GET = 1,
    DL_HP_OP_SET = 2,
    DL_HP_OP_CREATE = 3,
    DL_HP_OP_DELETE = 4,
    DL_HP_OP_TEST_SET = 5,
    DL_HP_OP_SEARCH = 6
} dl_hp_op_t

DL_HP_OP_GET Get information from HP-DLPI.
DL_HP_OP_SET Set a property in HP-DLPI.
DL_HP_OP_CREATE Register with HP-DLPI.
DL_HP_OP_DELETE Delete a driver instance.
DL_HP_OP_TEST_SET Not supported. Reserved for future use.
DL_HP_OP_SEARCH Not supported. Reserved for future use.

SEE ALSO

dlpi_propp(9F)
NAME
dl_hp_pkt_type_t - Packet types supported.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
This enumeration defines all packet types. HP-DLPI supports only a subset of the packet types defined here.

STRUCTURE MEMBERS
typedef enum dl_hp_pkt_type {
    DL_HP_ETHER_PKT = 0,
    DL_HP_IEEE8023XSAP_PKT = 1,
    DL_HP_SNAP8023_PKT = 2,
    DL_HP_ETHERXT_PKT = 3,
    DL_HP_SNAP8024_PKT = 4,
    DL_HP_IEEE8023_PKT = 5,
    DL_HP_SNAP8023XT_PKT = 6,
    DL_HP_SNAP8024XT_PKT = 7,
    DL_HP_SNAPFDDI_PKT = 8,
    DL_HP_SNAP8025_PKT = 9,
    DL_HP_IEEE8025_PKT = 10,
    DL_HP_ATR_PKT = 11,
    DL_HP_FDDI_UI_PKT = 12,
    DL_HP_SNAPFDDI_LLA_PKT = 13,
    DL_HP_FDDI_LLA_PKT = 14,
    DL_HP_FCSNAP_PKT = 15,
    DL_HP_FC8022_PKT = 16,
    DL_HP_IEEE8025XSAP_PKT = 17,
    DL_HP_FDDIXSAP_PKT = 18,
    DL_HP_FDDI_PKT = 253,
    DL_HP_UNKNOWN_PKT = 254,
    DL_HP_RAW_PKT = 255
} dl_hp_pkt_type_t;

HP-DLPI supports the following packet types:

DL_HP_ETHER_PKT Ethernet packet type.

+-----------------+-----------------+----------+
| DA | SA | TYPE | PAYLOAD |
+-----------------+-----------------+----------+
<-----------------|-----------------|---------->
6 bytes | 6 bytes | 2 bytes

DL_HP_IEEE8023XSAP_PKT IEEE 802.3 Extended SAP packet type.

+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
| L | E | D | S | C | D | S | A |
| N | S | S | T | X | X | Y |
| G | A | A | R | HDR_FILL | S | S | L |
| T | P | P | L | A | A | O |
DL_HP_SNAP8023_PKT  SNAP 802.3 packet type.

DL_HP_IEEE8023_PKT  IEEE 802.3 packet type.

DL_HP_RAW_PKT  Raw HP-DLPI packet. Drivers must not modify the packet in any form.

SEE ALSO

dlpi_propp(9F)
NAME

dl_hp_promisc_state_t - Promiscuous mode information that is passed for the DLPI_PROP_PROMISC_INFO property.

SYNOPSIS

#include <sio/dlpi_drv.h>

DESCRIPTION

This enumerated type is used during DLPI_PROP_PROMISC_INFO property call to get the current promiscuous mode information for an interface. This is a bit wise enumeration that will return all the promiscuous levels currently enabled.

STRUCTURE MEMBERS

typedef enum dl_hp_promisc_state {
    DL_HP_PHYSICAL_LEVEL   = 0x1,
    DL_HP_SAP_LEVEL        = 0x2,
    DL_HP_MULTI_LEVEL      = 0x4
} dl_hp_promisc_state_t;

DL_HP_PHYSICAL_LEVEL          Physical level promiscuous mode is enabled.
DL_HP_SAP_LEVEL               SAP level promiscuous mode is enabled.
DL_HP_MULTI_LEVEL             Multicast level promiscuous mode enabled.

SEE ALSO

dlpi_propp(9F)
dl_hp_prop_t(9S)

NAME

dl_hp_prop_t - List of HP-DLPI properties for a driver.

SYNOPSIS

#include <sio/dlpi_drv.h>

DESCRIPTION

This enumeration type contains the properties that are handled by HP-DLPI. The properties listed here are for tightly-coupled and loosely-coupled drivers. The properties are passed as part of the dlpi_propp call to HP-DLPI. Each property is explained along with the values HP-DLPI expects for dlpi_hdlp, valuep, and opcode when calling dlpi_propp.

STRUCTURE MEMBERS

typedef enum dl_hp_prop {
   DLPI_PROP_VERSION,
   DLPI_PROP_DRV_INSTANCEP,
   DLPI_PROP_WAKEUPP,
   DLPI_PROP_EVENTP,
   DLPI_PROP_HDW_STATE,
   DLPI_PROP_MAC_ADDR,
   DLPI_PROP_RESERVED1,
   DLPI_PROP_INBOUNDP,
   DLPI_PROP_NMID,
   DLPI_PROP_FEATURES_ONE,
   DLPI_PROP_IFADMIN,
   DLPI_PROP_MCAST_INFOP,
   DLPI_PROP_PROMISC_INFO,
   DLPI_PROP_PROTOCOL_INFOP,
   DLPI_PROP_VLAN_INFOP,
   DLPI_PROP_OOP_INFOP,
   DLPI_PROP_RESERVED1,
   DLPI_PROP_TRAP_OPTZER,
   DLPI_PROP_IFLASTCHANGE,
   DLPI_PROP_DRIVER_INFOP,
   DLPI_PROP_MODULE_INFOP,
   DLPI_PROP_SPECIAL_PARAMS
} dl_hp_prop_t;
**DLPI_PROP_VERSION**

dlpi_hdlp: Can be set to NULL, if a HP-DLPI provided handle is not available.

opcode: DL_HP_OP_GET

valuep:

**In:** Pointer to uint32_t

**Out:** The pointer content is set to the HP-DLPI version number.

This property is valid for tightly coupled and loosely coupled drivers. HP-DLPI recommends the drivers to get this property as the first step before registering its instance with HP-DLPI. Getting this property will help drivers know what version of DLPI they are working with and to see if the drivers and DLPI are compatible.

After getting this property, the driver must set the value received in dhc_version element of the dl_hp_create_info_t structure during create operation. The dhc_version field is the version of HP-DLPI with which the driver is compatible.

The HP-DLPI version number will be set to 2 for the HP-UX 11i v2.

HP-DLPI Version 2 will not be compatible with future HP-DLPI versions.

**DLPI_PROP_DRV_INSTANCEP**

dlpi_hdlp: Must be set to NULL.

opcode: DL_HP_OP_CREATE

valuep:

**In:** Pointer to dl_hp_create_info_t.

**Out:** Pointer to the same structure with dhc_dlpi_hdlp field of dl_hp_create_info_t filled with pointer to the HP-DLPI private handle.

Description:

This property is valid for tightly coupled and loosely coupled drivers. This property is provided to accomplish the Create operation of a driver instance. In addition, it allows drivers to register some of their properties with the HP-DLPI.

**DLPI_PROP_WAKEUPP**

dlpi_hdlp: May be set to NULL, if a HP-DLPI provided handle is not available

opcode: DL_HP_OP_GET

valuep:

**In:** Pointer to HP-DLPI wakeup routine. See dlpi_suspend(9F) for more information.

**Out:** Filled with a function pointer.

Description:

This property is valid only for tightly coupled drivers. This property is used to get the pointer to the HP-DLPI function. The HP-DLPI wakeup function must be called on completion of all control requests.
All control requests passed via the driver_control_interface are assumed to be asynchronous in nature and require the HP-DLPI wakeup function to be called on completion of the request.

**DLPI_PROP_EVENTP**

dlpi_hdlp: Can be set to NULL, if a HP-DLPI provided handle is not available

opcode: DL_HP_OP_GET

valuep:

**In:** Pointer to HP-DLPI event handler routine. See dlpi_eventp(9F) for more information.

**Out:** Filled with a function pointer.

This property is valid only for tightly coupled drivers.

This property is used to get the pointer to the HP-DLPI event handler. The HP-DLPI event handler function can be used to send HP-DLPI defined events triggered by drivers.

**DLPI_PROP_HDW_STATE**

dlpi_hdlp: Must be the HP-DLPI provided handle.

opcode: DL_HP_OP_SET

valuep:

**In:** Pointer to a variable of type dl_hp_hw_state_t.

This property is valid for tightly coupled and loosely coupled drivers. Drivers must call dlpi_propp with this property to update changes in the hardware state of the interface.

Driver must generate link DOWN and UP events if the hardware state of the driver instance changes. See the HP-UX 11i v3 Driver Development Guide for more information.

Tightly coupled drivers must call this property after the initialization completes. Until such a time, the driver instance will not be visible to normal users. Drivers must also update HP-DLPI with correct operating parameters (for example, MAC address and MTU) before making this property call.

**DLPI_PROP_MAC_ADDR**

dlpi_hdlp: Must be the HP-DLPI provided handle.

opcode: DL_HP_OP_SET

valuep:

**In:** Pointer to uint8_t.

This property is valid for tightly coupled and loosely coupled drivers.

This property is used to communicate changes in the MAC address of the interface to HP-DLPI. This property must be called by drivers whenever the interface MAC address changes.

The MAC address must be in canonical format for Ethernet and FDDI and in wire format for Token Ring. The length of the MAC address must be same as the value specified.
by the driver during DL_HP_OP_CREATE in the dhc_mac_addr_len field of the dl_hp_create_info_t structure.

Tightly coupled drivers must generate events if the MAC address of the driver instance changes. See the HP-UX 11i v3 Driver Development Guide for more information.

**DLPI_PROP_RESERVED1**

Reserved by HP-DLPI.

**DLPI_PROP_INBOUNDP**

dlpi_hdlp: Must be the HP-DLPI provided handle.

*opcode:* DL_HP_OP_GET

*valuep:*

*In:* Pointer to the HP-DLPI inbound processing routine. See `dlpi_inboundp(9F)` for more information.

*Out:* Filled with a function pointer.

This property is valid only for tightly coupled drivers. Tightly coupled drivers can invoke this property, any time after successful registration of a driver instance. This property is used to get the pointer to the HP-DLPI inbound routine, which has to be invoked for frames coming in on that interface.

**DLPI_PROP_NMID**

dlpi_hdlp: Must be the HP-DLPI provided handle.

*opcode:* DL_HP_OP_GET/DL_HP_OP_SET

*valuep:*

For DL_HP_OP_GET:

*In:* Pointer to `uint32_t`.

*Out:* Filled with NMID.

For DL_HP_OP_SET:

*In:* Pointer to `uint32_t`, filled with NMID to be set for the interface.

*Get/Set* the NM ID of the interface.

This property is valid for tightly coupled and loosely coupled drivers. However, only loosely coupled drivers are permitted to modify NMID.

For tightly coupled drivers, the NMID is allocated by HP-DLPI, and tightly coupled drivers can get it from HP-DLPI using this property.

**DLPI_PROP_FEATURES_ONE**

dlpi_hdlp: Must be the HP-DLPI provided handle.

*opcode:* DL_HP_OP_SET

*valuep:*

*In:* Pointer to `dl_hp_drv_features_one_t`.

This property allows tightly coupled drivers to set or change the features supported by drivers.

Driver must generate events if the features of the driver instance changes. See the HP-UX 11i v3 Driver Development Guide for more information.
DLPI_PROP_IFADMIN

`opcode`: DL_HP_OP_SET

`valuep`:

**In**: Pointer to dl_hp_ifadmin_state_t.

This property is valid for tightly coupled drivers.

Tightly coupled drivers must call this property to set or change the `ifAdmin` value of the interface.

Driver must generate events if the `ifAdmin` state of the driver instance changes. See the HP-UX 11i v3 Driver Development Guide for more information.

DLPI_PROP_MCAST_INFOP

`dlpi_hdlp`: Must be the HP-DLPI provided handle.

`opcode`: DL_HP_OP_GET

`valuep`:

**In**: Pointer to dl_hp_getinfo_t.

**Out**: Fills up the `dl_hp_getinfo_t` structure and sets the `dhg_datap` element to point to an MBLK containing the list of unique multicast addresses enabled on the interface.

The `MBLK->b_rptr` will point to the first multicast address in the list of multicast address returned.

This property is valid for tightly coupled drivers. This property can be used to get the list of unique multicast addresses enabled on a instance and the multicast addresses themselves.

HP-DLPI will allocate the `dhg_datap` element of `dl_hp_getinfo_t` structure.

The `dhg_datap` element of the `dl_hp_getinfo_t` structure must be freed using the `dhg_free_funcp` routine provided in the same structure. No locks must be held across the call to `dhg_free_funcp`.

DLPI_PROP_PROMISC_INFO

`dlpi_hdlp`: Must be the HP-DLPI provided handle.

`opcode`: DL_HP_OP_GET

`valuep`:

**In**: Pointer to dl_hp_promisc_state_t.

**Out**: Filled with promiscuous level. HP-DLPI will return a bitmask containing all the levels that are enabled.

This property is valid for tightly coupled drivers.

Get the promiscuous level that is expected to be enabled on the card. HP-DLPI provides the promiscuous level on the instance by taking into account:

- All the promiscuous requests from all streams attached to the interface.
- The promiscuous level implied when the number of unique multicast enabled on the interface exceeds the NIC/driver specified limit.

DLPI_PROP_PROTOCOL_INFOP

Reserved for future use.
**DLPI_PROP_VLAN_INFOP**

- **dlpi_hdlp**: Must be the handle provided by HP-DLPI.
- **opcode**: DL_HP_OP_GET
- **valuep**:
  - **In**: Pointer to dl_hp_getinfo_t.
  - **Out**: Fills up the dl_hp_getinfo_t structure and sets the dhg_datap element to point to a memory area containing an array of VLANs created on the interface. Each element of the array is of type dl_hp_vlan_instance_info_t.

  This property is valid for tightly coupled drivers. This property can be used to get the list of VLANs enabled on an instance whose HP-DLPI handle is dlpi_hdlp.

  HP-DLPI will allocate the dhg_datap element of the dl_hp_getinfo_t structure.

  The dhg_datap element of the dl_hp_getinfo_t structure must be freed using the dhg_free_funcp routine provided in the same structure. No locks must be held across the call to dhg_free_funcp.

**DLPI_PROP_OOP_INFOP**

- **dlpi_hdlp**: Must be the HP-DLPI provided handle.
- **opcode**: DL_HP_OP_GET
- **valuep**:
  - **In**: Pointer to dl_hp_oop_template_t.
  - **Out**: Filled structure of type dl_hp_oop_template_t.

  HP-DLPI will set the dhot_mblkp element of the structure with the OOP template.

  This property is valid only for tightly coupled drivers.

  This property must be used by drivers that support out-of-packet data on the inbound side. See dl_hp_drv_features_one_t(9F) for information on which drivers must use the DLPI_PROP_OOP_INFOP. HP-DLPI will calculate the maximum out-of-packet data size which the driver must to use to allocate additional size for each inbound packet and also the template of the OOP data.

  The driver can provide the padding space required in the OOP template to avoid IP pull-ups (4 byte aligned IP header).

  The MBLK provided here by HP-DLPI must not be freed by the driver at any time.

  If a driver supports outbound OOP but does not support inbound OOP, for loopback packets the driver needs to move the mblk read pointer to point to the actual payload before looping back the packets.

  See the HP-UX 11i v3 Driver Development Guide for information on how to implement OOP in a HP-DLPI based network interface driver.

**DLPI_PROP_INSTANCE_INFOP**

- **dlpi_hdlp**: Must be NULL.
- **opcode**: DL_HP_OP_GET
valuep:

In: Pointer to dl_hp_getinfo_t.

Out: Filled structure of type dl_hp_getinfo_t. HP-DLPI will set the dhg_datap element of dl_hp_getinfo_t to a memory area that contains multiple entries of type dl_hp_instance_info_t.

Search and get interface information about all interfaces that match the criteria specified in the dhg_modifier field of the dl_hp_getinfo_t structure passed via the valuep parameter.

On a successful return, the dhg_datap field of the dl_hp_getinfo_t structure passed in valuep will contain information about all the ports that match the criteria.

The information present in dhg_datap will be an array of type dl_hp_instance_info_t.

<table>
<thead>
<tr>
<th>DLPI_PROP_MODULEP</th>
<th>Reserved for internal use.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLPI_PROP_MOD_ACQUIRE_PORT</td>
<td>Reserved for internal use.</td>
</tr>
<tr>
<td>DLPI_PROP_MOD_RESERVED</td>
<td>Reserved for internal use.</td>
</tr>
<tr>
<td>DLPI_PROP_MOD_RELEASE_PORT</td>
<td>Reserved for internal use.</td>
</tr>
<tr>
<td>DLPI_PROP_MOD_ASSOCIATE_PORT</td>
<td>Reserved for Internal use.</td>
</tr>
<tr>
<td>DLPI_PROP_MOD_DISASSOCIATE_PORT</td>
<td>Reserved for internal use.</td>
</tr>
<tr>
<td>DLPI_PROP_MOD_UPDATE_INSTANCEP</td>
<td>Reserved for internal use.</td>
</tr>
<tr>
<td>DLPI_PROP_MOD_CONTROL_LP</td>
<td>Reserved for internal use.</td>
</tr>
<tr>
<td>DLPI_PROP_RESERVED4</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>DLPI_PROP_RESERVED2</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>DLPI_PROP_OP_PARAM</td>
<td>dlpi_hdlp: Must be the HP-DLPI provided per-interface handle.</td>
</tr>
</tbody>
</table>

opcode: DL_HP_OP_SET

valuep:

In: Pointer to dl_hp_op_param_t.

Description:

This property is valid for tightly coupled and loosely coupled drivers.

This property will set any one or more of the following values: Speed, Duplex, autoneg/auto sense/MTU.

The speed must be given in Megabits per second (Mb/s).

The MTU must be given in bytes. The minimum MTU must be 10; the minimum recommended MTU is 256.

Duplex value is of type dl_hp_duplex_mode_t.

autoneg/Autosense value is of type dl_hp_autoneg_sense_t.

Driver must generate events if any of the operating parameters giving in this structure change. See the HP-UX 11i v3 Driver Development Guide for more information.
DLPI_PROP_OOP_OUT_INFOP

*dlpi_hdlp*: Must be an HP-DLPI provided per-interface handle.

*opcode*: DL_HP_OP_SET

*valuep*:

**In**: Pointer to *dl_hp_oop_out_template_t*.

This property is valid only for tightly coupled drivers.

This property must be used by the drivers that expect upper layers to pass additional OOP information that HP-DLPI does not understand. The template that is passed as part of this property is included in the outbound template that HP-DLPI creates for HP-DLPI users.

DLPI_PROP_RESERVED3

Reserved for internal use.

DLPI_PROP_TARP_OPTZER

Reserved for internal use.

DLPI_PROP_IFLASTCHANGE

*dlpi_hdlp*: Must be the HP-DLPI provided handle.

*opcode*: DL_HP_OP_GET

*valuep*:

**In**: Pointer to *TimeTicks*.

**Out**: Filled with a time value in increments of 10 milliseconds.

This property is for tightly coupled drivers.

This property must be used by drivers to set its internal ifLastChange element of the MIB II structure. This value must be directly assigned to ifLastChange. Drivers must set ifLastChange whenever there is a change in its ifOper state.

DLPI_PROP_DRIVER_INFOP

Reserved for internal use.

DLPI_PROP_MODULE_INFOP

Reserved for internal use.

DLPI_PROP_SPECIAL_PARAMS

*dlpi_hdlp*: Must be the HP-DLPI provided handle.

*opcode*: DL_HP_OP_GET/DL_HP_OP_SET

*valuep*:

**In**: *dl_hp_special_params_t*

**Out**:

For DL_HP_OP_SET:

**In**: Pointer filled with *dl_hp_special_params_t*.

This property is valid only for tightly coupled drivers.

Tightly coupled drivers must call this property in order to get and set special parameters associated with the interface.

SEE ALSO

*dl_hp_special_params_t*(9S)
NAME
DL_HP_RESET_STATS_REQ - Request to reset physical interface statistics.

SYNOPSIS
#include <sys/dlpi_ext.h>

DESCRIPTION
This primitive is sent to drivers using the driver_controlp routine to reset the physical
interface statistics. The associated structure for this request is the dl_hp_reset_stats_req_t
structure.

The DL_HP_RESET_STATS_REQ primitive described in this manpage might be different from
the definition in the DLPI Programmer’s Guide. The information provided here is applicable only
to tightly coupled drivers.

STRUCTURE MEMBERS
typedef struct {
    uint32_t dl_primitive;
} dl_hp_reset_stats_req_t;

dl_primitive  Set to DL_HP_RESET_STATS_REQ

CONSTRAINTS
Drivers must call HP-DLPI event handler with DL_HP_EVENT_CLEAR_STATS event to reset
per-instance HP-DLPI level statistics after resetting its internal statistics. See dlpi_eventp(9F) for
more information.

SEE ALSO
driver_controlp(9F), dlpi_eventp(9F)
NAME

dl_hp_search_info_t - Search criteria to search the interface list.

SYNOPSIS

#include <sio/dlpi_drv.h>

DESCRIPTION

Specification for searching the interface list. Used when invoking the
DLPI_PROP_INSTANCE_INFOP properties.

STRUCTURE MEMBERS

typedef struct dl_hp_search_info {
    dl_hp_search_type_t dhsi_match;
    uint32_t dhsi_ppa;
    char *dhsi_drv_namep;
    dl_hp_drv_features_one_t dhsi_features;
    uint32_t dhsi_features_two;
    uint32_t dhsi_features_three;
    dl_hp_mac_type_t dhsi_mac_type;
    void *dhsi_rsvdp;
} dl_hp_search_info_t;

dhsi_match Specifications which of the following fields are to be matched during
the search of the global interface list. Multiple fields could be
specified and the search will match all the specified criteria. See
dl_hp_search_type_t(9F) for more information.

dhsi_ppa PPA to search for.

dhsi_drv_namep Get all instances of this driver that matches the driver name passed.
The name must be less than or equal to 64 characters and must be
NULL terminated. The name is case sensitive.

dhsi_features Match all the interfaces that have the same set or a super-set of
these features. See dl_hp_drv_features_one_t(9F) for more
information.

dhsi_features_two Reserved for future use. Must be set to 0.

dhsi_features_three Reserved for future use. Must be set to 0.

dhsi_mac_type Match all the interfaces that have the MAC type given. See
dl_hp_mac_type_t(9F) for additional information.

dhsi_rsvdp Reserved for future use. Must be set to 0.

SEE ALSO

dlpi_propp(9F), dl_hp_features_one_t(9S), dl_hp_mac_type_t(9S)
dl_hp_search_type_t(9S)

NAME
dl_hp_search_type_t - Criteria for searching the interface list.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
Specifications for searching the interface list. This is a bit wise enumerated type.

STRUCTURE MEMBERS

typedef enum dl_hp_search_type {
    DL_HP_MATCH_PPA,
    DL_HP_MATCH_DRIVER_NAME,
    DL_HP_MATCH_FEATURES,
    DL_HP_MATCH_MAC_TYPE
} dl_hp_search_type_t

DL_HP_MATCH_PPA Match the PPA given in the dhsi_ppa element of dl_hp_search_info_t structure.
DL_HP_MATCH_DRIVER_NAME Match the driver name given in the dhsi_drv_namep element of dl_hp_search_info_t structure.
DL_HP_MATCH_FEATURES Match the features given in the dhsi_features element of dl_hp_search_info_t structure.
DL_HP_MATCH_MAC_TYPE Match the MAC type given in the dhsi_mac_type element of dl_hp_search_info_t structure.

SEE ALSO
dl_hp_search_info_t(9S)
NAME

DL_HP_SET_DRV_PARAM_IOCTL - HP-DLPI defined transparent IOCTL to set driver parameters.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

This IOCTL is sent to drivers using the driver_controlp routine. The associated structure for this request is dl_hp_set_drv_param_ioctl_t structure.

This IOCTL is defined by HP-DLPI for usage by drivers. This IOCTL is transparent to HP-DLPI (there are few exceptions, see the CONSTRAINTS section). Transparent IOCTLs are the ones that are not interpreted by HP-DLPI and are sent directly to the driver.

Use this IOCTL to set driver operating parameters like speed, duplex mode, MTU, and autonegotiation. Drivers can add new sub-requests to satisfy any driver specific commands. See the HP-UX 11i v3 Driver Development Guide for more information on defining sub-requests.

STRUCTURE MEMBERS

typedef struct dl_hp_set_drv_param_ioctl {
    dl_hp_drv_param_req_type_t dl_request;
    uint32_t                   dl_speed;
    dl_hp_duplex_mode_t        dl_duplex;
    dl_hp_autoneg_sense_t      dl_autoneg;
    uint32_t                   dl_mtu;
    uint64_t                   dl_value1;
    uint64_t                   dl_value2;
    uint64_t                   dl_value3;
    uint32_t                   dl_reserved1[2];
    uint32_t                   dl_reserved2[2];
} dl_hp_set_drv_param_ioctl_t;

dl_request The sub-request. The driver property(s) that need to be set. See dl_hp_drv_param_req_type_t(9F) for more information.

dl_speed Set the speed of underlying link in Megabits per second (Mb/s).

dl_duplex Set full or half duplex modes. The field is of type dl_hp_duplex_mode_t.

dl_autoneg Set the auto-negotiation/auto-sense. The field is of type dl_hp_autoneg_sense_t.

dl_mtu Set the MTU of the interface in octets

dl_value1 64-bit value for driver extension.

dl_value2 64-bit value for driver extension.

dl_value3 64-bit value for driver extension.


dl_reserved2[2] Reserved for internal use by HP-DLPI. Must be set to 0.

CONSTRAINTS

HP-DLPI does not interpret this request (it is transparent) except for the DL_HP_SERIALIZE bit. This request must be used between the driver's user space application and the driver.
The driver user space component can request HP-DLPI to serialize this request when sending it to the driver, if another DL_HP_SET_DRV_PARAM_IOCTL on the same interface is in progress, by setting the DL_HP_SERIALIZE bit in the dl_request field. If serialization is requested, DLPI will wait for the in-progress DL_HP_SET_DRV_PARAM_IOCTL to complete before sending another.

The drivers can define their own dl_request by using any of the bits reserved for driver's extension or defining new bits that are not defined by HP-DLPI. The reserved bits are defined in dl_hp_drv_param_req_type_t (9F).

The driver must check the privileges of the user who issued the request (using the ioc_cr field of iocblk).

The driver must fill up the ioc_count value in the IOCTL MBLK if the request is successfully processed.

If the IOCTL fails, HP-DLPI will free the b_cont of the MBLK that contains the data associated with the IOCTL.

**SEE ALSO**

dlpi_controlp (9F), dl_hp_drv_param_req_type_t (9S), dl_hp_duplex_mode_t (9S).

dl_hp_autoneg_sense_t (9S)
NAME

DL_HP_SET_IFADMIN_REQ - Request to set the physical interface administrative status.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

This primitive is sent to drivers using the driver_controlp routine to set the administrative state of the interface. The associated structure for this request is dl_hp_set_ifadmin_req_t structure.

This primitive is applicable only between HP-DLPI and the driver.

STRUCTURE MEMBERS

typedef struct dl_hp_set_ifadmin_req {
    uint32_t             dl_primitive;
    dl_hp_ifadmin_state_t dl_ifAdminStatus;
    uint32_t             dl_reserved;
} dl_hp_set_ifadmin_req_t;

dl_primitive Set to DL_HP_SET_IFADMIN_REQ.
dl_ifAdminStatus The ifAdmin value to be set on the interface. The element is of type dl_hp_ifadmin_state_t.
dl_reserved Reserved for future use. Set to 0.

CONSTRAINTS

Requests driver to set the administrative status of the interface to the value specified in dl_ifAdminStatus.

Tightly coupled drivers must generate link DOWN and UP events when the ifAdmin status changes. See the HP-UX 11i v3 Driver Development Guide for more information.

SEE ALSO

driver_controlp(9F), dl_hp_ifadmin_state_t(9S)
dl_hp_special_params_t(9S)

NAME
dl_hp_special_params_t - Used to get or set special driver parameters.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
Specification to get or set driver special parameters. Used when invoking the
DLPI_PROP_SPECIAL_PARAMS properties.

STRUCTURE MEMBERS
typedef struct dl_hp_special_params {
    dl_hp_sp_param_type_t dsp_param_type;
    dl_hp_sp_params_t     dsp_params;
} dl_hp_special_params_t;

dsp_param_type   Specifies the type of special parameters that the driver supports. For
                 example, TCP segmentation offload.

dsp_params      Driver special parameters for a set or get operation.

SEE ALSO
dlpipropp(9F), dl_hp_sp_param_type(9S), dl_hp_special_params(9S)
dl_hp_sp_param_type(9S)

NAME
dl_hp_sp_param_type - Enumeration used to get or set special driver parameters.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
Specification to get or set driver special parameters. Used when invoking the
DLPI_PROP_SPECIAL_PARAMS properties.

STRUCTURE MEMBERS
typedef enum dl_hp_sp_param_type {
    DL_HP_TCP_SEG
} dl_hp_sp_param_type_t;

DL_HP_TCP_SEG Specifies the type of special parameters that the driver supports.

DL_HP_TCP_SEG for TCP segmentation offload is currently the only special
parameter type that is supported.

SEE ALSO
dlpi_propp(9F), dl_hp_special_params_t(9S), dl_hp_sp_params_t(9S)
dl_hp_sp_params_t(9S)

NAME
dl_hp_sp_params_t - A union used to get or set special driver parameters.

SYNOPSIS
#include <sio/dlpi_drv.h>

DESCRIPTION
Specification to get or set driver special parameters. Used when invoking the
DLPI_PROP_SPECIAL_PARAMS specification to get or set driver special parameters. Used when
invoking the DLPI_PROP_SPECIAL_PARAMS properties.

STRUCTURE MEMBERS
typedef union dl_hp_sp_params {
    dl_hp_get_tcp_seg_info_t dsp_tcpseg_info;
    uint64_t                 dsp_reserved[2];
} dl_hp_sp_params_t;

dsp_tcpseg_info  Data related to TCP Segmentation Offload is a type of special parameter
                 supported by some drivers. This contains information like virtual MTU,
                 segmentation flags, and minimum number of segments to be created
                 from original segment.

dsp_reserved     Reserved for future use.

SEE ALSO
dl_hp_sp_param_type(9S), dl_hp_special_params(9S), dl_hp_tcpseg_info_t(9S)
**NAME**

dl_hp_src_llc_info_t - Exchange source information during LLC build header routine.

**SYNOPSIS**

```c
#include <sio/dlpi_drv.h>
```

**DESCRIPTION**

This structure is used by HP-DLPI to pass source information to the driver build header routine.

**STRUCTURE MEMBERS**

```c
typedef struct dl_hp_src_llc_info {
    uint8_t       dsli_ssap;
    uint16_t      dsli_sxsap;
    uint32_t      dsli_dlsap_addr_length;
    uint8_t       dsli_dlsap_addr[DL_HP_MAX_DLSAP_LEN];
    dl_hp_tagging_type_t dsli_tagging_type;
    uint32_t      dsli_pkt_length;
    uint32_t      dsli_ip_stream;
    uint32_t      dsli_arp_stream;
    uint32_t      dsli_no_src_routing;
    uint8_t       dsli_ctrl;
    uint64_t      dsli_reserved1[2];
    uint64_t      dsli_reserved2[2];
} dl_hp_src_llc_info_t;
```

- **dsli_ssap**
  Identifies the source SAP. Depending on the type of packet (XID/TEST Response or request) the LSB of the SAP value will be set. The build header routine must use the SAP value from this element instead of extracting the value from `dsli_dlsap_addr`.

- **dsli_sxsap**
  Identifies the extended source SAP. You can extract this value from `dsli_dlsap_addr` as well.

- **dsli_dlsap_addr_length**
  Identifies the DLSAP address length in bytes.

- **dsli_dlsap_addr**
  Identifies the source DLSAP address. See the *HP-UX 11i v3 Driver Development Guide* for more information.

- **dsli_tagging_type**
  Identifies whether the VLAN packet is for a driver that supports software-based VLAN tagging or hardware-based VLAN tagging. For an enumeration of supported tagging types, see `dl_hp_tagging_type_t(9S)`.

- **dsli_pkt_length**
  Identifies the packet length in bytes. This is the data length of the packet (excluding the LLC header length). Drivers must add to this length field before filling the length for IEEE802.3 and SNAP802.3 packets.

- **dsli_ip_stream**
  Set to 1 if the packet is an IPv4 or IPv6 packet. This helps the driver's build header routine in understanding the format of RIF information.

- **dsli_arp_stream**
  Set to 1 if the packet is an ARP packet. This helps the driver's build header routine in understanding the format of RIF information.
dsli_no_src_routing

Set to 1 if Source routing is disabled (only for Token Ring packets).

Identifies the control type.

Reserved for future use. It must be set to 0.

Reserved for future use. It must be set to 0.

SEE ALSO
dl_hp_tagging_type_t(9S)
NAME

dl_hp_tagging_type_t - List of VLAN tagging types that drivers can support.

SYNOPSIS

#include <sio/dlpi_ext.h>

DESCRIPTION

List of VLAN tagging types that drivers can support. Used by HP-DLPI to pass source information to the driver build header.

STRUCTURE MEMBERS

typedef enum dl_hp_tagging_type {
    DL_HP_INTERFACE_PHYSICAL = 1,
    DL_HP_INTERFACE_SWVLAN = 2,
} dl_hp_tagging_type_t;

DL_HP_INTERFACE_PHYSICAL Set if VLAN interface is created over a device capable of hardware tagging.

DL_HP_INTERFACE_SWVLAN Set if VLAN interface is created over a device capable of software tagging.

SEE ALSO

dl_hp_src_llc_info_t(9S)
DL_HP_USAGE_INFO_ACK(9G)

NAME

DL_HP_USAGE_INFO_ACK - Response to DL_HP_USAGE_INFO_REQ request.

SYNOPSIS

#include <sys/dlpi_ext.h>

DESCRIPTION

This primitive is used to return a response for DL_HP_USAGE_INFO_REQ request. Driver-specific usage information follows this primitive. The driver is responsible for allocating the buffer for the acknowledgement and usage information. An application, such as NetCRA, can also use this primitive to get the usage of an interface.

The response information is present in the dl_hp_usage_info_ack_t structure.

STRUCTURE MEMBERS

typedef struct {
    uint32_t       dl_primitive;
    uint32_t       dl_length;
    uint32_t       dl_count;
    uint32_t       dl_offset;
    uint32_t       dl_reserved1;
    uint32_t       dl_reserved2;
} dl_hp_usage_info_ack_t;

dl_primitive Set to DL_HP_USAGE_INFO_ACK.

dl_length Specifies the data length, in bytes, in the acknowledgement, not including dl_hp_usage_info_ack_t itself. Note dl_length might not be the same as dl_count multiplied by sizeof(dl_hp_usage_info_t).

dl_count Specifies the number of dl_hp_usage_info_t usage information structures.

dl_offset Specifies the offset from the beginning of the acknowledgement to the first dl_hp_usage_info_t structure.

dl_reserved1 Reserved for future use. Must be set to 0.

dl_reserved2 Reserved for future use. Must be set to 0.

CONSTRAINTS

The mblk that contains the acknowledgement must be a M_PCPROTO type.

SEE ALSO

driver_controlp(9F), DL_HP_USAGE_INFO_REQ(9F), dl_hp_usage_info_t(9S)
**NAME**

DL_HP_USAGE_INFO_REQ - Request to retrieve usage information for a specific interface.

**SYNOPSIS**

```c
#include <sys/dlpi_ext.h>
```

**DESCRIPTION**

This primitive is conditionally sent to drivers using the `driver_controlp` routine to retrieve driver-specific usage information. Only drivers that set the `DL_HP_DRV_USAGE_INFO` feature bit will receive this request.

In addition to the information returned from drivers, DLPI also incorporates all usages it knows about, including VLANs created over the interface, and links them as a list using `dl_next_offset` field of `dl_hp_usage_info_t` data structure.

The request information is present in the `dl_hp_usage_info_req_t` structure.

**STRUCTURE MEMBERS**

```c
typedef struct {
    uint32_t       dl_primitive;
    uint32_t       dl_reserved1;
    uint32_t       dl_reserved2;
} dl_hp_usage_info_req_t;
```

- `dl_primitive` Set to `DL_HP_USAGE_INFO_REQ`.
- `dl_reserved1` Reserved for future use. Must be set to 0.
- `dl_reserved2` Reserved for future use. Must be set to 0.

**CONSTRAINTS**

If the driver sets the feature bit to handle this primitive, the driver must allocate the buffer, which must be of type `mblk_t` type. The buffer contains two parts: `dl_hp_usage_info_ack_t` and a list of `dl_hp_usage_info_t` data structures, with offsets in each to get to the next element in the list. Each entry contains one application, which is directly accessing the driver and bypassing DLPI.

In the case where no usage information is returned, the driver must still allocate a buffer for the `dl_hp_usage_info_ack_t`, and set the `dl_length` field and `dl_count` to zero. (See `DL_HP_USAGE_INFO_ACK(9F)` for more information).

The response can be in multiple `mblk` structures. However, the `dl_next_offset` is calculated as if the buffers are continuous (or one buffer). The `dl_next_offset` of last entry must be set to zero.

Any `dl_hp_usage_info_t` data structure must be always in one `mblk`.

The response `mblk` structures must be linked (through `b_cont`) with the request `mblk` before returning to the upper layer from the driver. The first `mblk`, which is the request `mblk`, will be freed by DLPI.

Any failure due to the memory allocation must be returned as error with status of `ENOBUFS`.

**SEE ALSO**

`driver_controlp(9F), dl_hp_usage_info_t(9S), DL_HP_USAGE_INFO_ACK(9F)`
NAME

dl_hp_usage_info_t - Structure containing one entry of usage information.

SYNOPSIS

#include <sio/dlpi_ext.h>

DESCRIPTION

This data type carries one usage information entry. Depending on the usage type, the dl_info field must be interpreted using different structure.

STRUCTURE MEMBERS

typedef struct {
    dl_hp_info_type_t dl_info_type
    dl_hp_info_t      dl_info;
    uint32_t          dl_next_offset;
} dl_hp_usage_info_t;

dl_info_type Specifies the type of information contained in the dl_info element. The dl_info_type is of type dl_hp_info_type_t.

dl_info Contains the information depending on dl_info_type.

dl_next_offset Specifies the offset from the beginning of the acknowledgement to the next dl_hp_usage_info_t data structure.

SEE ALSO

dl_hp_info_type_t(9S), dl_hp_info_t(9S), DL_HP_USAGE_INFO_ACK(9S)
NAME
dl_hp_vlan_info_t - Structure containing the VLAN information created in the underlying interface.

SYNOPSIS
#include <sio/dlpi_ext.h>

DESCRIPTION
This data structure is used to carry information for one VLAN interface that is created on the interface being analyzed. Note, only when the VLAN is directly created on the interface will information be reported.

STRUCTURE MEMBERS
typedef struct dl_hp_vlan_info {
    uint32_t  dl_vppa;
    uint32_t  dl_vlanid;
    uint32_t  dl_reserved1;
    uint32_t  dl_reserved2;
} dl_hp_vlan_info_t;

dl_vppa Specifies the Physical Point of Attachment (PPA) number of the VLAN interface.

dl_vlanid The VLAN identifier of the interface.

dl_reserved1 Reserved for future use. Must be set to 0.

dl_reserved2 Reserved for future use. Must be set to 0.

SEE ALSO
dl_hp_info_t(9S), dl_hp_info_type_t(9S), dl_hp_usage_info_t(9S)
dl_hp_vlan_tag_t(9S)

NAME
dl_hp_vlan_tag_t - Data type for the DL_HP_OOP_VLAN_TAG OOP type.

SYNOPSIS
#include <sio/dlpi_ext.h>

DESCRIPTION
Contains the VLAN tag passed by drivers that support hardware-based tagging to DLPI. This
OOP type is defined in dl_hp_oop_type_t.

STRUCTURE MEMBERS
typedef struct dl_hp_vlan_tag {
    uint16_t dhvt_tag;
    uint16_t dhvt_reserved;
} dl_hp_vlan_tag_t;

dhvt_tag          VLAN tag.
dhvt_reserved     Reserved for future use.

SEE ALSO
dl_hp_oop_type_t(9S)
NAME

DL_PHYS_ADDR_REQ - Request to get physical address (MAC address).

SYNOPSIS

#include <sys/dlpi.h>

DESCRIPTION

This primitive is sent to drivers using the `driver_controlp` routine to retrieve physical address of the card. The request could be to retrieve factory default address or the current physical address. The request information is present in the `dl_phys_addr_req_t` structure.

The `DL_PHYS_ADDR_REQ` primitive described in this manpage may be different from the definition in the `DLPI Programmer’s Guide`. The information provided here is applicable only to tightly coupled drivers.

STRUCTURE MEMBERS

```c
typedef struct {
    uint32_t dl_primitive;
    uint32_t dl_addr_type;
    uint32_t dl_addr_offset;
} dl_phys_addr_req_t;
```

dl_primitive Set to `DL_PHYS_ADDR_REQ`

dl_addr_type Type of physical address requested:

- `DL_FACT_PHYS_ADDR` — To retrieve factory default physical address.
- `DL_CURR_PHYS_ADDR` — To retrieve current physical address.

dl_addr_offset Offset from the beginning of the message block (`MBLK->b_rptr`) where the physical address begins.

CONSTRAINTS

The second MBLK (linked by `b_cont`) contains space to hold the physical address.

The second MBLK is empty when HP-DLPI passes it to the driver (`b_rptr == b_wptr`). After writing the information to the second MBLK, drivers must move the `b_wptr` to the size that was written to the second MBLK (in this case by the size of the physical address).

HP-DLPI handles MAC addresses in canonical format for Ethernet and FDDI drivers and wire format for Token Ring drivers.

SEE ALSO

`driver_controlp(9F)`
DL_PROMISCOFF_REQ(9G)

NAME

DL_PROMISCOFF_REQ - Request to disable a promiscuous level on the physical interface.

SYNOPSIS

#include <sys/dlpi.h>

DESCRIPTION

This primitive is sent to drivers using the driver_controlp routine to disable promiscuous mode on the physical interface. The associated structure for this request is the dl_promiscoff_req_t structure.

The DL_PROMISCOFF_REQ primitive described in this manpage may be different from the definition in the DLPI Programmer’s Guide. The information provided here is applicable only to tightly coupled drivers.

STRUCTURE MEMBERS

typedef struct {
    uint32_t dl_primitive;
    uint32_t dl_level;
} dl_promiscoff_req_t;

dl_primitive Set to DL_PROMISCOFF_REQ

CONSTRAINTS

The dl_level parameter must be ignored by the driver and must disable all the promiscuous levels enabled on the card.

HP-DLPI will always succeed the DL_PROMISCOFF_REQ request to the user/application, even if driver failed in its processing. If the driver fails the request, it will go out-of-sync (the promiscuous level in the driver will be different from what is maintained in HP-DLPI) with HP-DLPI. To recover from this situation, the driver can attempt to reprogram the promiscuous mode (by getting them from HP-DLPI using a property) at a convenient time. Failure to do so, will result in performance degradation.

If this primitive is received when the card is in physical or multicast promiscuous mode, the driver must disable all promiscuous levels. In addition, the driver must get the multicast list from HP-DLPI using the driver_propp call and reprogram the card.

SEE ALSO

driver_controlp(9F), dlpi_propp(9F)
NAME

DL_PROMISCON_REQ - Request to enable a promiscuous level on the physical interface.

SYNOPSIS

#include <sys/dlpi.h>

DESCRIPTION

This primitive is sent to drivers using the driver_controlp routine to enable a level of promiscuous mode on the interface. The associated structure for this request is the dl_promiscon_req_t structure.

The DL_PROMISCON_REQ primitive described in this manpage may be different from the definition in the DLPI Programmer's Guide. The information provided here is applicable only to tightly coupled drivers.

STRUCTURE MEMBERS

typedef struct {
   uint32_t dl_primitive;
   uint32_t dl_level;
} dl_promiscon_req_t;

dl_primitive Set to DL_PROMISCON_REQ

dl_level Indicates promiscuous level. The valid values for the promiscuous level are:

   DL_PROMISC_PHYS — Physical promiscuous mode.
   DL_PROMISC_SAP — SAP promiscuous mode.
   DL_PROMISC_MULTI — Multicast promiscuous mode.

CONSTRAINTS

If the HP-DLPI requested promiscuous level in dl_level is different from the level that is enabled in the card, the driver can disable the current level and enable the level requested by HP-DLPI.

If the promiscuous level changes to DL_PROMISC_SAP from a higher level (DL_PROMISC_MULTI or DL_PROMISC_PHYS), the driver must get the list of multicast addresses from HP-DLPI using driver_propp and reprogram the card.

HP-DLPI will send the DL_PROMISCON_REQ primitive only if a user has requested higher promiscuous level than the card is already in.

SEE ALSO

driver_controlp(9F), dlpi_propp(9F)
NAME

DL_SET_PHYS_ADDR_REQ - Request to set physical address (MAC address).

SYNOPSIS

#include <sys/dlpi.h>

DESCRIPTION

This primitive is sent to drivers using the driver_controlp routine to set the physical address of the card. The request is to change the physical interface address. The request information is present in the dl_set_phys_addr_req_t structure.

The DL_SET_PHYS_ADDR_REQ primitive described here might be different from the definition in the DLPI Programmer’s Guide. The information provided here is applicable only to tightly coupled drivers.

STRUCTURE MEMBERS

typedef struct {
    uint32_t    dl_primitive;
    uint32_t    dl_addr_length;
    uint32_t    dl_addr_offset;
} dl_phys_addr_req_t;

dl_primitive Set to DL_SET_PHYS_ADDR_REQ

dl_addr_length Length of the physical address.

dl_addr_offset Offset from the beginning of the message block (b_rptr of the message block) where the multicast address begins.

CONSTRAINTS

MAC address passed to the driver will in canonical format for Ethernet drivers.

Drivers need to generate link events when there is a change in physical address. See the HP-UX 11i v3 Driver Development Guide for more information on events.

SEE ALSO

driver_controlp(9F)
NAME
dlpi_eventp - HP-DLPI event handler.

SYNOPSIS
#include <sio/dlpi_drv.h>
void (*dlpi_eventp)(
    void *dlpi_hdlp,
    dl_hp_event_type_t event,
    void *event_infop,
    void *rsvdp
);  

PARAMETERS
dlpi_hdlp Pointer to HP-DLPI private handle (opaque to drivers) that was obtained
during initialization as part of the DL_HP_OP_CREATE operation.

event HP-DLPI event to process. The element is of type dl_hp_event_type_t.

event_infop Associated information for the event.

rsvdp Reserved for future use. Must be set to NULL.

DESCRIPTION
This is the HP-DLPI event handler. This interface is applicable only for tightly coupled drivers.
The event handler will be used to notify HP-DLPI of link events (such as state changes, clear
HP-DLPI's internal statistics, and so on). The event handler routine will take additional hints or
information, if required. For events like link DOWN and UP, drivers must pass the hint which
specifies why the event happened. For example, when driver sends a link DOWN event during
the MAC address change, it will pass the information in the event_infop parameter. For link UP
events, a driver can use DL_HP_EVENT_DEFAULT if the actual cause of link UP event is not
known (see dl_hp_event_link_cause_t(9F)).

RETURN VALUES
None

CONSTRAINTS
This routine must be requested from HP-DLPI using the driver_propp routine. The
DLPI_PROP_EVENTP property will return the function pointer to this routine (see
dl_hp_prop_t(9F)).

No locks must be held while calling this routine.

HP-DLPI does not guarantee the synchronous execution of events in the same thread, but
guarantees the order of processing of the events (the events will be processed in the order they
are received).

WARNINGS
None

EXAMPLES
SEE ALSO
dlpi_propp(9F), dl_hp_prop_t(9S), dl_hp_event_link_cause_t(9S), dl_hp_event_type_t(9S)
NAME

dlpi_inboundp - HP-DLPI inbound packet processing routine.

SYNOPSIS

#include <sio/dlpi_drv.h>

void (*dlpi_inboundp) (
    void *dlpi_hdlp,
    mblk_t *mblkp,
    void *rsvd1p,
    void *rsvd2p
) ;

PARAMETERS

dlpi_hdlp Pointer to HP-DLPI private handle (opaque to drivers) that was obtained during the DL_HP_OP_CREATE operation.

mblkp Inbound datagram.

rsvd1p Reserved for future use. Must be set to NULL.

rsvd1p Reserved for future use. Must be set to NULL.

DESCRIPTION

Inbound path for all packet formats. HP-DLPI provides inbound routines for Ethernet technologies.

RETURN VALUES

None.

CONSTRAINTS

The inbound routine must be requested from HP-DLPI using the dlpi_propp interface. The property to use is DLPI_PROP_INBOUNDP.

No locks must be held when calling the inbound routine.

The out-of-packet data template created by HP-DLPI during driver's call to dlpi_propp with DLPI_PROP_OOP_INFOP property must be used to pass any OOP information. See the HP-UX 11i v3 Driver Development Guide for HP-DLPI to Driver options negotiations and out-of-packet data passing mechanism.

The b_rptr of the MBLK must either point to the OOP if it is present or to the MAC header (that is, the destination address for Ethernet).

If OOP is present, the b_flag of the MBLK containing the OOP must have DL_HP_OOP_PRESENT flag set, see the HP-UX 11i v3 Driver Development Guide for more information.

The beginning of the MBLK data (b_rptr) must be 4 byte aligned.

When the driver is looping back an outbound packet, the driver must also pass the OOP information that was sent down by HP-DLPI.

Drivers that handle packet trains, must copy the cko_offset element of cko_info_t structure and the VLAN Tag information from the first fragment of the packet train to the subsequent fragment that is being looped back. The fragments must be separated and sent as individual packets to HP-DLPI. See the HP-UX 11i v3 Driver Development Guide for more information.

For loopback 802.3 packet trains, the drivers must also calculate the length of each one of the packet fragments and place it in the length part of LLC header before sending to HP-DLPI.
reason is when a packet train is sent to the driver, HP-DLPI will update the length part of the
802.2 header only for the first fragment of the packet train.
HP-DLPI handles automatic response for SAP 0 TEST packets.

**SEE ALSO**

dlpi_propp(9F)
NAME
dlpi_propp - HP-DLPI property interface

SYNOPSIS
#include <sio/dlpi_drv.h>

int32_t (*dlpi_propp)(
    void *dlpi_hdlp,
    dl_hp_op_t opcode,
    dl_hp_prop_t prop_name,
    void *valuep,
    void *rsvd1p,
    void *rsvd2p
);

PARAMETERS

dlpi_hdlp  Pointer to HP-DLPI private handle (opaque to drivers) that was obtained during
           the DL_HP_OP_CREATE operation (dlpi_hdlp must be set to NULL for the
           DL_HP_OP_CREATE operation itself). It is possible to pass NULL for dlpi_hdlp
           when calling dlpi_propp for certain properties. See dl_hp_prop_t(9F) for more
           information.

opcode    Enumerated type (dl_hp_op_t) specifying the operation.
           DL_HP_OP_CREATE   Create a new instance of HP-DLPI per-instance
                              structures.
           DL_HP_OP_DELETE   Delete data structures pertaining to this instance.
           DL_HP_OP_GET      Retrieve a property.
           DL_HP_OP_SET      Set a property in HP-DLPI.

Support for an opcode depends on the property to which it is applied.

prop_name Property to Create/Delete/Get/Set. Parameter is of type dl_hp_prop_t.

valuep    For DL_HP_OP_GET, this is an “in” and “out” parameter.
           For DL_HP_OP_SET, this is an “in” parameter. This parameter specifies the value
           associated with the property.

rsvd1p    Reserved for future use. Must be set to NULL.

rsvd2p    Reserved for future use. Must be set to NULL.

DESCRIPTION
This interface will be provided by HP-DLPI and must be used by all drivers to exchange
interface-specific information between HP-DLPI and the driver. In general, a pull approach is
favored; the module that needs the data pulls it from the module supplying the data by calling
the supplier’s property function. Exceptions, where they occur, are noted.

The property routine is a well-known global function pointer. The syntax of the function will
not change once a driver or module is written to use it.

RETURN VALUES
0        for success.
EBUSY    HP-DLPI cannot handle this property at this time.
EINVAL  Invalid property call. For example, loosely coupled drivers doing a get on the HP-DLPI event handler or the HP-DLPI inbound routine.
ENOBDFS  Allocation of message blocks failed.
ENOENT  No entries found.
ENOMEM  Memory allocation failure.
ENOTSUP  Unsupported property or operation.

**CONSTRAINTS**

No locks must be held while calling into this routine.
Call to *dlpi_propp* will not block or sleep.
Caller must pass the buffer for the structure associated with a *DL_HP_OP_GET* property. Any additional allocations required to satisfy the request is allocated by HP-DLPI and must be freed by the driver using the free routine returned along with the information returned. There are few exceptions, see *dl_hp_prop_t* (9F) for the individual property definitions.
HP-DLPI private handle must be passed for all the *dlpi_propp* calls (see *dl_hp_prop_t* (9F) for exceptions). The *DL_HP_OP_CREATE* operation will return the HP-DLPI private handle to the caller.
The *DL_HP_OP_DELETE* operation for tightly-coupled physical drivers is supported only when the initialization of the driver instance fails. After initialization, the driver instance cannot be deleted. Deletion of loosely-coupled driver instances is allowed anytime.
HP-DLPI maintains a copy of some of the driver properties like Speed, MTU, ifAdmin status for the purpose of giving it to other modules that depend on them. Such properties must be set by the driver using the list of *DL_HP_OP_SET* properties listed in *dl_hp_prop_t* (9F). The driver is responsible for maintaining the copy of driver properties in HP-DLPI.
HP-DLPI recommends that property calls be avoided in performance paths as they are not performance-friendly.
If a property call fails, all the actions that are supposed to happen in that property has failed. In other words, no partial processing of a property call is supported.
The values associated with the properties can be categorized into “In” and “Out” values depending on whether the value is passed to HP-DLPI by the caller or HP-DLPI returns the value to the callee. “In” means that content of the *valuep* parameter is passed by the driver to HP-DLPI. “Out” means the value is passed back from HP-DLPI to the driver. “In/Out” means, the information can be passed in by the driver while calling HP-DLPI and, HP-DLPI can pass back information when the call returns.

**SEE ALSO**

*dl_hp_prop_t* (9S)
NAME
dlpi_wakeupp - HP-DLPI interface to be used by drivers to notify the asynchronous completion of control requests.

SYNOPSIS
#include <sio/dlpi_drv.h>
void (*dlpi_wakeupp)(
    void *dlpi_hdlp,
    mblk_t *mblkp,
    uint32_t status,
    void *rsvdp
);

PARAMETERS
dlpi_hdlp  Pointer to HP-DLPI private handle (opaque to drivers) that was obtained during initialization as part of the DL_HP_OP_CREATE operation.
mblkp     The address of the MBLK containing the request that was passed to the driver (see driver_controlp(9F)).
status    Status of the control request for which the driver is doing the wakeup.
rsvdp     Reserved for future use. Must be set to NULL.

DESCRIPTION
HP-DLPI provides a wakeup handler for tightly coupled drivers to do an asynchronous notification of the completion of a control request HP-DLPI sent to the driver. All the requests sent to HP-DLPI using driver_controlp will require a wakeup. Drivers will be able to return errors for all the control requests in the wakeup handler routine.

RETURN VALUES
None

CONSTRAINTS
This routine must be requested from HP-DLPI during the initial property exchange. The property to use is DLPI_PROP_WAKEUPP (see dl_hp_prop_t(9F)).
No locks must be held when calling this routine.
Driver must guarantee that all the control requests are woken up. Failure to do so may cause subsequent control requests to be blocked by DLPI.
The drivers must not issue redundant wakeup for a control request. Even though HP-DLPI has mechanisms to detect this kind of situation, it may potentially cause erroneous behavior.
The dlpi_wakeupp call may happen before the driver_controlp call completes, but driver must not access the control request MBLK after the wakeup call has been made.

SEE ALSO
driver_controlp(9F), dlpi_wakeupp(9F), driver_controlp(9F)
driver_build_headerp(9F)

NAME

driver_build_headerp - Driver packet build header routine.

SYNOPSIS

#include <sio/dlpi_drv.h>

uint32_t (*driver_build_headerp)(
    void *driver_hdlp,
    dl_hp_src_llc_info_t *source_infop,
    dl_hp_dest_llc_info_t *dest_infop,
    dl_hp_llc_info_t *llc_hdrp,
    void *rsvd1p,
    void *rsvd2p
);

PARAMETERS

driver_hdlp Pointer to opaque instance specific control handle given to HP-DLPI by the
    driver during initialization. HP-DLPI passes this to driver when calling
driver. This is typically driver's interface structure.

source_infop This contains the source information for the packet including source SAP,
    source address, source extended SAP, VLAN information, and RIF
    information. This is used to construct the LLC header. This element will be
    of type dl_hp_src_llc_info_t.

dest_infop The destination information, such as dsap, dxsap, and destination
    addr. Parameter is of type dl_hp_dest_llc_info_t.

llc_hdrp Pointer to the structure containing the LLC header information. HP-DLPI
    will allocate this parameter. The dli_mblkp element of dl_hp_llc_info_t
    structure is allocated and built by the build header routine. The structure
    will be of type dl_hp_llc_info_t.

rsvd1p Reserved for future use. Set to NULL.

rsvd2p Reserved for future use. Set to NULL.

DESCRIPTION

The purpose of the build header routine is to create the MAC and LLC header for outbound
packets. This routine will also be called during fastpath header template negotiations (see the
HP-UX 11i v3 Driver Development Guide) with transport to create the MAC and LLC header
template. HP-DLPI will provide a default build header routine for Ethernet drivers if they do
not provide their own build header routine. This driver build header entry point is provided by
driver during the property exchange time (see dlpi_propp(9F)) as part of the DL_HP_OP_CREATE
operation. If the routine provided is NULL, HP-DLPI uses HP-DLPI's build header routine.

RETURN VALUES

0 Success.

EINVAL Header cannot be created because of invalid values in elements of source_infop and
    dest_infop have incorrectly formatted. Set llc_hdrp to NULL.

ENOBUFFS Failure to allocate message buffers.

CONSTRAINTS

No locks will be held by HP-DLPI when calling the build header routine.
This interface is optional. HP-DLPI will use its internal build header routine to construct Ethernet packet headers.

This interface may be provided by the driver to HP-DLPI during property exchange as dhc_build_hdrp element of dl_hp_create_info_t structure (see dlpi_propp(9F)).

If the driver provides a build header routine it will be requested to build the link level header. HP-DLPI will handle MAC and multicast addresses in canonical format only for Ethernet drivers.

If the driver build header routine finds an error in the information passed or the build header processing is not successful, it must return NULL in the dli_mblkp element of dl_hp_llc_info_t pointer and must return an error value for this routine.

Drivers are not allowed to block or sleep during this call.

**SEE ALSO**

dlpi_propp(9F), dl_hp_src_llc_info_t(9S), dl_hp_dest_llc_info_t(9S), dl_hp_llc_info_t(9S), dl_hp_create_info_t(9S)
driver_controlp(9F)

NAME

driver_controlp - Driver control request handler.

SYNOPSIS

#include <sio/dlpi_drv.h>

void (*driver_controlp)(
    void *driver_hdlp,
    int32_t cmd,
    mblk_t *mblkp,
    void *rsvdp
);

PARAMETERS

driver_hdlp  Pointer to opaque instance specific control handle given to HP-DLPI by the
driver during initialization as dhc_control_hdlp in
dl_hp_create_info_t. HP-DLPI passes this handle to the driver when
calling driver's control routine. This is typically a driver's per-instance
structure.

cmd  A 32-bit integer command/request value.

mblkp  Pointer to MBLK containing the data associated with the request. See the
DESCRIPTION section for the list of requests that will be passed to the driver.
The request MBLK passed must not be modified by the driver.

rsvdp  Reserved for future use. Set to NULL.

DESCRIPTION

Tightly coupled drivers provide a control request handler that the HP-DLPI module calls to pass
any requests to the drivers. Drivers, for all the control requests irrespective of whether they
support or not, must do a wakeup using the dlpi_wakeupp interface. See the HP-UX 11i v3
Driver Development Guide for more information on HP-DLPI's simulated sleep/wakeup mechanism
and the control requests and kinds of information that will be passed along with the control
requests.

For certain requests, HP-DLPI will allocate the required size to hold the information that is
requested from the driver and link it to the b_cont of the request MBLK passed. The driver will
copy the required information depending on the request and pass it back during the wakeup of
HP-DLPI by calling the wakeup function. HP-DLPI will then do rest of the processing of
constructing the acknowledgement MBLK before responding to the user. The IOCTLs are
transparent to HP-DLPI. HP-DLPI passes the IOCTLs down to the driver and expects the driver
to do all the processing, including allocation of additional space.

HP-DLPI requests contain the request information in the first message block and optionally a
second MBLK linked by b_cont of the request MBLK. The second MBLK is used to pass
information from the driver to HP-DLPI for requests such as DL_GET_STATISTICS_REQ.

IOCTLs requests are generally transparent to HP-DLPI, except for
DL_HP_SET_DRV_PARAM_IOCTL. See DL_HP_SET_DRV_PARAM_IOCTL(9F) for more
information). IOCTLs when passed to the driver will have first MBLK containing the iocblk
structure, which provides information to the driver such as the IOCTL command and the user
credentials. Following the first MBLK, linked by b_cont, is a second MBLK that contains any
IOCTL specific information.

The list of requests that will be passed to the driver are:

DL_ENABMULTI_REQ  Request to enable specific multicast addresses.
<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL_DISABMULTI_REQ</td>
<td>Request to disable specific multicast addresses.</td>
</tr>
<tr>
<td>DL_PHYS_ADDR_REQ</td>
<td>Request to get the physical address.</td>
</tr>
<tr>
<td>DL_SET_PHYS_ADDR_REQ</td>
<td>Set the physical address to the value passed in this request.</td>
</tr>
<tr>
<td>DL_HP_HW_RESET_REQ</td>
<td>Request to reset the physical interface.</td>
</tr>
<tr>
<td>DL_PROMISCOFF_REQ</td>
<td>Request to disable promiscuous mode.</td>
</tr>
<tr>
<td>DL_PROMISCON_REQ</td>
<td>Request to enable promiscuous mode.</td>
</tr>
<tr>
<td>DL_HP_RESET_STATS_REQ</td>
<td>Request to reset the MIB statistics of an interface.</td>
</tr>
<tr>
<td>DL_GET_STATISTICS_REQ</td>
<td>Request to get the standard and extended MIB statistics.</td>
</tr>
<tr>
<td>DL_HP_GET_MIBSTATS_REQ</td>
<td>Request to get the standard and extended MIB statistics applicable to all interfaces.</td>
</tr>
<tr>
<td>DL_HP_GET_64BIT_STATS_REQ</td>
<td>Request to get the 64-bit standard and extended MIB statistics applicable to those interfaces that support 64-bit statistics.</td>
</tr>
<tr>
<td>DL_HP_SET_IFADMIN_REQ</td>
<td>Set the ifAdmin value.</td>
</tr>
<tr>
<td>DL_HP_CREATE_VLAN_REQ</td>
<td>Request to create a VLAN on an interface.</td>
</tr>
<tr>
<td>DL_HP_DELETE_VLAN_REQ</td>
<td>Request to delete a VLAN on an interface.</td>
</tr>
<tr>
<td>DL_HP_MODIFY_VLAN_REQ</td>
<td>Request to modify properties of a VLAN interface.</td>
</tr>
<tr>
<td>DL_HP_SET_DRV_PARAM_IOCTL</td>
<td>HP-DLPI defined transparent IOCTL to set driver parameters.</td>
</tr>
<tr>
<td>DL_HP_GET_DRV_PARAM_IOCTL</td>
<td>HP-DLPI defined transparent IOCTL to get driver parameters.</td>
</tr>
</tbody>
</table>

See the respective manpages for additional information.

For an illustration of the flow of processing for these requests, see the *HP-UX 11i v3 Driver Development Guide*.

**RETURN VALUES**

None

**CONSTRAINTS**

All control requests received by this routine must be woken up. A missed wakeup call by the driver may result in future control requests being blocked. See *dlpi_wakeupp(9F)* for the wakeup processing routine.

Multiple control requests can be sent to the driver at the same time. Driver must take care of queueing and handling them. Drivers may choose to return error if they receive a control request while handling another request, but care must be taken to retain the request MBLK to wakeup HP-DLPI with error.

If the control request is to change the property of the interface (for example, MTU, speed, and MAC address), the drivers must inform HP-DLPI of the change using the *dlpi_propp* interface and may have to generate link DOWN and UP events. To find out how and when to generate link DOWN and UP events, see the *HP-UX 11i v3 Driver Development Guide*.

No locks will be held by HP-DLPI while calling this routine.

This interface must be provided by the driver to HP-DLPI during property exchange as *dhc_controlp* element of *dl_hp_create_info_t* structure.

Driver specific application and drivers can define IOCTLs that will be transparent to HP-DLPI. Transparent IOCTLs will be passed directly to the driver without any processing in HP-DLPI. For constraints on defining transparent IOCTLs, see the *HP-UX 11i v3 Driver Development Guide*. For the definitions of IOCTLs, see the individual manpages.
HP-DLPI will allocate space for driver to fill the information for requests that require driver to pass information back to DLPI (for example, DL_GET_STATISTICS_REQ), see the individual request for this information.

See dlpi_propp(9F) for an explanation of driver control interface handle that will be passed as first parameter with every invocation of dlpi_controlp routine.

Drivers must not modify the control request MBLK (the first MBLK) in any way.

Drivers must not modify the b_rptr of the second MBLK (if there is a second MBLK).

Drivers are not allowed to block or sleep during this call.

For IOCTLs driver must do the necessary credential checks.

**WARNINGS**

None

**EXAMPLES**

**SEE ALSO**

dlpi_prop(9F), dlpi_wakeup(9F), dl_hp_create_info_t(9S), driver_controlp(9F),
DL_ENABMULTI_REQ(9G), DL_DISABMULTI_REQ(9G), DL_SET_PHYS_ADDR_REQ(9G),
DL_PHYS_ADDR_REQ(9G), DL_HP_HW_RESET_REQ(9G), DL_PROMISCOFF_REQ(9G),
DL_PROMISCON_REQ(9G), DL_HP_RESET_STATS_REQ(9G), DL_GET_STATISTICS_REQ(9G),
DL_HP_GET_MIBSTATS_REQ(9G), DL_HP_GET_64BIT_STATS_REQ(9G),
DL_HP_SET_IFADMIN_REQ(9G), DL_HP_CREATE_VLAN_REQ(9G),
DL_HP_DELETE_VLAN_REQ(9G), DL_HP_MODIFY_VLAN_REQ(9G),
DL_HP_SET_DRV_PARAM_IOCTL(9G), DL_HP_GET_DRV_PARAM_IOCTL(9G)
NAME

driver_event_handlerp - Driver event handler routine.

SYNOPSIS

#include <sio/dlpi_drv.h>

int32_t (*driver_event_handlerp)(
    void *driver_hdlp,
    dl_hp_drv_event_type_t event_type,
    void *event_infop,
    void *rsvdp
);

PARAMETERS

*driver_hdlp Pointer to opaque instance specific control handle given to HP-DLPI by the
driver during initialization. HP-DLPI passes this to driver when calling driver.
This is typically driver's interface structure.
*event_type Event Type. See dl_hp_drv_event_type_t(9F).
*event_infop Any supplement information associated with this event.
*rsvdp Reserved for future use. Set to NULL.

DESCRIPTION

This interface serves as an entry point into tightly coupled physical drivers to initiate some
process in the driver. This event handler is by HP-DLPI.

RETURN VALUES

0 Success.
ENXIO No such device.
ENOMEM Not enough kernel memory.
EINVAL One of the parameter has invalid value.
ENOBUFS Not enough buffer space (message block).
ENOLINK Interface is not connected.
ENOTSUP Unsupported event.

CONSTRAINTS

This interface must be provided by the driver to HP-DLPI during property exchange as
dhc_eventp element of dl_hp_create_info_t structure.
No locks will be held by HP-DLPI while calling this routine.
See dlpi_propp(9F) for an explanation of the driver event interface handle that will be passed as
first parameter with every invocation of driver_event_handlerp routine.
Drivers are not allowed to block (sleep) during this call.
Any information passed via event_infop parameter will be invalid after the driver returns from
the call. Drivers must copy the contents of event_infop if they need to reference them later.
WARNINGS
EXAMPLES
SEE ALSO
dlpi_propp(9F), dl_hp_drv_event_type_t(9S), driver_event_handlerp(9F), dl_hp_create_info_t(9S)
driver_outputp(9F)

NAME

driver_outputp - Driver outbound packet processing routine.

SYNOPSIS

#include <sio/dlpi_drv.h>

uint32_t (*driver_outputp)(
    void *driver_hdlp,
    mblk_t *mblkp,
    dl_hp_pkt_t pkt_type,
    void *rsvdp
    );

PARAMETERS

driver_hdlp Pointer to opaque instance specific control handle given to HP-DLPI by the
driver during initialization. HP-DLPI passes this to driver when calling driver.
This is typically driver's interface structure

mblkp Pointer to the data.

pkt_type Packet type. See dl_hp_pkt_type_t(9S).

rsvdp Reserved for future use. Set to NULL.

DESCRIPTION

This is the entry point for the drivers' data path. The driver output routine must be registered
with HP-DLPI during the driver property exchange (see dlpi_propp(9F)).
This routine applies only to tightly coupled drivers.

RETURN VALUES

0 Successfully enqueued the packet to the hardware transmit queue.
ENXIO No such device.
ENOMEM Not enough kernel memory.
EINVAL One of the parameter has invalid value.
ENOBUFFS Not enough buffer space (message block).
ENOLINK Interface is not connected.

CONSTRAINTS

Drivers must check for out-of-packet data by checking the b_flag bit for DL_HP_OOP_PRESENT
and process them. If the drivers want to ignore the OOP data, they can jump to the MAC/LLC
header based on the 4-byte offset (aligned at a word boundary) field that will be present in the
beginning of the packet. The 4-byte offset to the MAC/LLC header will be present only if
DL_OOP_PRESENT flag is set in the b_flag field. See the HP-UX 11i v3 Driver Development Guide
for more information on OOP data.
No locks will be held by HP-DLPI while calling this routine.
This interface must be provided by the driver to HP-DLPI during property exchange as
dhc_outputp element of dl_hp_create_info_t structure.
If drivers receive out-of-packet data, they must not assume the order of the data. They must
follow the guidelines described in the HP-UX 11i v3 Driver Development Guide.
Multicast, broadcast packets must be looped back if the MSGNOLOOP flag is not set in b_flag of
the MBLK. The packets must be looped back by doing the copymsg.
If promiscuous mode is enabled, the packets must be looped back irrespective of the MSGNOLOOP flag.

If driver loops back packets, it must update its inbound statistic counters.

HP-DLPI will not do any packet size checks against the driver MTU.

Packets received with \texttt{pkt\_type} of DL\_HP\_RAW\_PKT must not be modified by the driver.

See \textit{dlpi\_propp}(9F) for an explanation of the driver output interface handle that will be passed as first parameter with every invocation of the \texttt{driver\_outputp} routine.

HP-DLPI will handle MAC and multicast addresses in canonical format for Ethernet and FDDI drivers and in wire format for Token Ring drivers.

Driver must not change the structure and order of the OOP data in the packets when the packets are looped-back.

Drivers that handle packet trains, must copy the \texttt{cko\_offset} element of \texttt{cko\_info\_t} structure and the VLAN Tag information from the first fragment of the packet train to the subsequent fragment that is being looped back. See the \textit{HP-UX 11i v3 Driver Development Guide} for more information.

The beginning of a new fragment in a packet train is specified by setting of the DL\_HP\_OOP\_PRESENT flag in the \texttt{b\_flag} of the MBLK.

Drivers must not change the Source Address of the packet.

Drivers are not allowed to block or sleep during this call.

\textbf{WARNINGS}

None

\textbf{EXAMPLES}

\textbf{SEE ALSO}

\textit{dlpi\_propp}(9F), \textit{dl\_hp\_pkt\_type\_t}(9S), \textit{dl\_hp\_create\_info\_t}(9S), \textit{copymsg}(9F), \textit{driver\_outputp}(9F)
NAME
filter_packet - Checks if the packet matches filter criteria set up.

SYNOPSIS
#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>
int filter_packet(void);

PARAMETERS
None

DESCRIPTION
For the current packet, checks all the filters set to see if the packet needs to be formatted. The
subformatter must call this routine after calling one of set_up_xxx routines, which sets up the
globals.

RETURN VALUES
0 Packet must not be formatted.
<> 0 Packet needs to be formatted.

CONSTRAINTS
None

SEE ALSO
set_up_8022(9F), set_up_ether(9F), set_up_link(9F), set_up_ip(9F), format_link_nice(9F), format_link_raw(9F), format_link_terse(9F)
NAME

format_link_nice - Formats the link level packet using nice formatting.

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>

int format_link_nice(
    tl_msg_hdr_type *hdr,
    u_char *buffer,
    int len,
    char *linktype,
    char *line1,
    char *addlinfo,
    char *upperinfo
);

PARAMETERS

hdr  Pointer to the standard nettl message header.
 buffer Pointer to data beginning at the 802.2 level. The upper layer routines typically will not format data straight from this buffer, but the uppermost layers may display data at an appropriate offset into the buffer.
 len  Length of the buffer (including 802.2, excluding any lower layer data).
 linktype  String describing the type of link this information is carried over (for example, FDDI, 802.5, ETHER, 802.3).
 line1  Short string (less than 23 bytes) giving more information to be displayed on the same line as the source address. For example, "TYPE: 0x800" for Ethernet packets (NOT SNAP) or "LENGTH: 26" for 802.3 packets. You can leave this blank by passing "".
 addlinfo  Additional lines of information pertaining to data in the MAC header. (Blank for 802.3 and Ethernet, but could include formatted flags or other information in the MAC header for other link types). Must be terminated with a newline (\n).
 upperinfo  Other lines of information pertaining to data beyond the MAC header. Will be displayed only if the packet does not have 802.2 or Ethernet information present, that is, as in conjunction with set_up_link). Ordinarily must be left blank. If present, you may wish to include a separator.

DESCRIPTION

This routine formats the link level packet displaying all the upper layer information. This is the most detailed level of formatting and the subformatter must call this routine when nice formatting mode is enabled (the mode of formatting active is conveyed through ss_N_fmt_flag_type structure while calling subsys_N_format).

RETURN VALUES

0  Error in formatting
 <> 0  Packet successfully formatted.

CONSTRAINTS

None
SEE ALSO

format_packet(9F), format_link_raw(9F), format_link_terse(9F), set_up_8022(9F), set_up_ether(9F),
set_up_link(9F), set_up_ip(9F), subsys_N_format(9F)
NAME

format_link_raw - Formats the link level packet using raw formatting.

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>

int format_link_raw(
    tl_msg_hdr_type *hdr,
    u_char *buffer,
    int len,
    int offset,
    char *linktype,
    char *interface,
    char *line3,
    char *addlinfo
);

PARAMETERS

hdr       Pointer to the standard nettl message header.
buffer    Pointer to entire traced packet (including MAC). Use the offset parameter to control where the data actually begins printing.
len       Length of entire buffer.
offset    Offset to actually begin displaying the data; that is, if the MAC information is not to be shown. 802.3 and Ethernet do not display until the beginning of the 802.2 information or the Ethernet data (because the Source and Dest information are formatted out).
linktype  String describing the type of link this information is carried over, such as FDDI, 802.3, 802.5, or Ethernet.
interface String appended to the device ID and printed out in the interface=[interface] field. Pass a NULL pointer to suppress displaying any interface information.
line3     Short string, less than 14 bytes, giving information to be displayed on the same line as the addresses. For example, "Type: 0x800" for Ethernet packets (not SNAP) or "Length: 00-1a" for 802.3 packets. You can leave this blank by passing "".
addlinfo  (Blank for 802.3 and Ethernet, but may include formatted flags or other information in the MAC header for other link types). Terminated with a newline (\n).

DESCRIPTION

This routine formats the link level packet displaying all the upper layer information as hex/ASCII data. The subformatter must call this routine when raw formatting mode is enabled (the mode of formatting active is conveyed through ss_N_fmt_flag_type structure while calling subsys_N_format).

RETURN VALUES

0       Error in formatting.
<> 0     Packet successfully formatted.
CONTRAINTS
None

SEE ALSO

format_packet(9F), format_link_nice(9F), format_link_terse(9F), set_up_8022(9F), set_up_ether(9F),
set_up_link(9F), set_up_ip(9F), subsys_N_format(9F)
NAME
format_link_terse - Formats the link level packet using terse formatting.

SYNOPSIS
#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>
int format_link_terse(
    tl_msg_hdr_type *hdr,
    u_char *buffer,
    int len,
    char *linktype,
    char *line3,
    char *addlinfo
  );

PARAMETERS
hdr                Pointer to the standard nettl message header.
buffer             Pointer to data beginning at the 802.2 level. The upper layer routines typically will
                    not format data straight from this buffer.
len                 Length of the buffer (including 802.2, excluding any lower layer data).
linktype            String describing the type of link this information is carried over, plus any other
                    MAC layer information appropriate for terse mode. For 802.3 the linktype is simply
                    "8"; for Ethernet (not SNAP) it is "E".
addlinfo            String giving other MAC or upper layer information to be displayed (blank for
                    802.3 and Ethernet).

DESCRIPTION
This routine formats the link level packet displaying the upper layer information in a single line. This is the least detailed level of formatting and the subformatter must call this routine when terse formatting mode is enabled (the mode of formatting active is conveyed through ss_N_fmt_flag_type structure while calling subsys_N_format).

RETURN VALUES
0           Error in formatting.
<> 0       Packet successfully formatted.

CONSTRAINTS
None

SEE ALSO
format_packet(9F), format_link_nice(9F), format_link_raw(9F), set_up_8022(9F), set_up_ether(9F),
set_up_link(9F), set_up_ip(9F), subsys_N_format(9F)
NAME

get_opt_parms_type - Structure containing subsystem options processing information.

SYNOPSIS

#include<fmt.h>

DESCRIPTION

The get_opt_parms_type structure contains all the information required by the subformatter to process the options file. This structure is passed while calling the subsys_N_get_options function.

Structure

The get_opt_parms_type structure defined in fmt.h has the following fields:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>status_ptr</td>
<td>int *</td>
</tr>
<tr>
<td>subsys_strm</td>
<td>FILE *</td>
</tr>
<tr>
<td>error_strm</td>
<td>FILE *</td>
</tr>
<tr>
<td>log_strm</td>
<td>FILE *</td>
</tr>
<tr>
<td>ss_id</td>
<td>int</td>
</tr>
<tr>
<td>ss_name</td>
<td>char *</td>
</tr>
<tr>
<td>ss_msg_cat</td>
<td>nl_catd</td>
</tr>
<tr>
<td>ss_n_get_opt_flag</td>
<td>get_opt_flag_type</td>
</tr>
<tr>
<td>ss_options_ptr</td>
<td>char **</td>
</tr>
<tr>
<td>ss_output_fd</td>
<td>int</td>
</tr>
<tr>
<td>options_file_name</td>
<td>char *</td>
</tr>
<tr>
<td>options_filename_printed</td>
<td>int *</td>
</tr>
</tbody>
</table>

status_ptr

Contains the error code of the routine, if the returned value is -1. The <fmt.h> file gives a complete list of the error codes:

- FMTERR_INV_FLAGS  Invalid ss_n_get_opt_flag parameter.
- FMTERR_NO_MEM     There is not enough memory; a call to malloc failed.
- FMTERR_INV_OPT_FD Invalid options file descriptor passed in.
- FMT_INV_MC_FD     Invalid message catalog file descriptor passed in.
- FMTERR_SYS_ERROR  An error has been returned from a system call.

subsys_strm

FILE pointer to the file that refers to the temporary file containing the options specifically for the N subsystem.
This file is created by the caller prior to invoking `subsys_N_options` routine, and each line has been converted to lower case. All comments, blank lines and lines for other subsystems are already removed. In addition, the keyword identifying this subsystem has been stripped off each line, so only the options for this particular subsystem are in the file. Due to a special encoding of line number and other data, the `tl_get_line` routine must be used to get option lines from this stream file.

`error_strm`  
FILE pointer to the file that will receive error messages.

`log_strm`  
FILE pointer to the file that will receive a summary of all options and files in effect for the subsystem, generated by `subsys_N_get_options` routine. The `nettl` command reports the contents of this file after all the subsystems have finished reading their respective filter command files.

`ss_id`  
Subsystem ID number for the subsystem as found in the configuration file.

`ss_name`  
Subsystem name for the subsystem as found in the configuration file.

`ss_msg_cat`  
File descriptor pointing to the message catalog for the subsystem as found in the configuration file.

`ss_n_get_opt_flag`  
Type of flag is defined in `fmt.h` as:

```c
typedef struct {
    u_int trace_log_bit: 1;
    u_int parse_only_bit: 1;
    u_int reserved: 30;
} get_opt_flag_type;
```

`trace_log_bit`  
This flag is not needed and must not be used by `subsys_N_get_options`.

`parse_only_bit`  
This flag is set when the `subsys_N_get_options` routine does not need to process the information in the file, only parse the input and check for syntax and semantic errors.

`ss_options_ptr`  
Pointer to a pointer to a data structure containing the specific information processed by the `subsys_N_get_options` routine and passed on to the `subsys_N_format` routine to handle special formatting. This structure must be allocated and initialized by `subsys_N_get_options` routine.

`ss_output_fd`  
File descriptor referring to the file receiving the formatter output.

`options_file_name`  
A character string which contains the file name of the filter file passed to `netfmt` command with the `-c` option. The file name can be used in error and warning messages produced by `subsys_N_get_options` function while parsing the filter field. Subsequent messages need not display the file name.
Flag indicating if the file name needs to be printed in an error or warning message. A value of 0 indicates that the file name must be printed.

**SEE ALSO**

`subsys_N_get_options(9F)`
NAME
kget_log_instance - Produces a unique number for use as a log instance.

SYNOPSIS
#include <net_diag.h>
unsigned short kget_log_instance(void);

PARAMETERS
None

DESCRIPTION
The kget_log_instance network function returns a unique number tied to the specific instance of a networking interface. This number is for use as a log instance value, for threading log messages together so all messages for the same instance can be identified together. A change in the log instance means that a new event is being logged.

The log instance value must be passed between subsystems through their interface parameter list, so each module may access it. If a module encounters a unique event, it obtains a log instance value. Otherwise, the module uses the current log instance value it obtained earlier through a call to kget_log_instance.

RETURN VALUES
n  A unique number for use as a log instance value.

CONSTRAINTS
None

SEE ALSO
klogg_write(9F)
KLOG_CK(9F)

NAME

KLOG_CK - Checks whether logging is enabled for the current subsystem.

SYNOPSIS

#include <sys/net_diag.h>
#include <sys/subsys_id.h>

int KLOG_CK(
    int subsys_id,
    int log_class
);  

PARAMETERS

subsys_id  The unique subsystem ID of the calling subsystem (number assigned by Hewlett-Packard).

log_class  Defines the classification of event. All classes are defined in the header file <sys/subsys_id.h>. Four classes are defined for logging messages:

INFORMATIVE  Normal messages only.

WARNING  Warning messages.

ERROR  Error condition messages.

DISASTER  Critical error messages

DESCRIPTION

The KLOG_CK network service is a macro that allows the calling process to find out whether logging is enabled for the current subsystem (given by subsys_id) and class (given by log_class).

RETURN VALUES

0  Logging is disabled.

1  Logging is enabled.

CONSTRAINTS

None

SEE ALSO

klogg_write(9F)
NAME

klogg_write - Sends log messages to the kernel trace and log facility.

SYNOPSIS

#include <sys/net_diag.h>
#include <sys/subsys_id.h>

int klogg_write(
    short subsys_id,
    int device_id,
    u_short log_instance,
    caddr_t tl_packet,
    int tl_packet_cnt
);

DESCRIPTION

The klogg_write network function sends log messages to the kernel trace and log facility. Prefiltering is done at the time of the log call, and unwanted messages are dropped.

PARAMETERS

subsys_id The unique ID (number assigned by Hewlett-Packard) of the calling subsystem.

class The classification of event. All classes are defined in the header file <sys/subsys_id.h>. Four classes are defined for logging messages:

INFORMATIVE Normal messages only.

WARNING Warning messages.

ERROR Error condition messages.

DISASTER Critical error messages.

device_id The device ID number (for example, enet driver’s instance number) of the calling subsystem message. If this is a non-applicable parameter, pass in -1.

log_instance A unique static number used to identify the thread of events attending an interface. If this is a non-applicable parameter, pass in -1.

tl_packet Either a pointer to an mblk chain or a pointer to a set of iovec structures as determined by tl_packet_cnt. This structure is immediately copied into memory allocated by the tracing and logging facilities, so the calling routine need not copy the data and then pass a pointer to the data.

tl_packet_cnt If -2, tl_packet points to an mblk chain. If the value is greater than 0, it is the number of iovec structures (as defined in <sys/uio.h>) that tl_packet points to.

RETURN VALUES

This routine always returns a 0.

CONSTRAINTS

None

SEE ALSO

KLOG_CK(9F)
KTRC_CHECK(9F)

NAME

KTRC_CHECK - Checks whether tracing is enabled; verifies tracing activation on a per-interface basis.

SYNOPSIS

#include <sys/net_diag.h>
#include <sys/subsys_id.h>

int KTRC_CHECK(int subsys_id, int trace_kind, int device_id);

PARAMETERS

subsys_id The unique subsystem ID of the calling subsystem (number assigned by Hewlett-Packard).

trace_kind The kind of trace. Available kinds are defined in the <sys/subsys_id.h> header file as follows:

ERROR_TRACE_BIT Error tracing mask.
HDR_IN_BIT Inbound header tracing mask.
HDR_OUT_BIT Outbound header tracing mask.
LOGGING_TRACE_BIT Log call tracing mask.
LOOP_BACK_BIT For loopback.
PDU_IN_BIT Inbound PDU tracing mask.
PDU_OUT_BIT Outbound PDU tracing mask.
PROCEDURE_TRACE_BIT Procedure entry/exit trace.
PTOP_BIT For point to point.

device_id The device ID number (for example, enet driver’s instance number). It can be used for filtering on a per-interface basis.

DESCRIPTION

The KTRC_CHECK network service is a macro that verifies whether tracing is enabled for the current subsystem (given by subsys_id), kind (given by trace_kind), and interface (given by device_id).

In a system with more than one interface card installed, this macro allows tracing on a per-interface basis, thereby reducing the impact of tracing on performance.

RETURN VALUES

0 Tracing is disabled.
1 Tracing is enabled.

CONSTRAINTS

None

EXAMPLES

The enet driver might use this macro as follows to check if tracing is enabled on the current instance for outbound data.
if (KTRC_CHECK(ENET_ID, PDU_OUT_BIT,
    (enet_iftp->lancift.hwift.instance_num))
    }
    ktrc_write(...);

SEE ALSO

ktrc_write(9F), KTRC_CK(9F)
KTRC_CK(9F)

NAME
KTRC_CK - Checks whether tracing is enabled.

SYNOPSIS
#include <sys/net_diag.h>
#include <sys/subsys_id.h>

int KTRC_CK(
    int subsys_id,
    int trace_kind
);

PARAMETERS
subsys_id The unique subsystem ID of the calling subsystem (number assigned by Hewlett-Packard).
trace_kind The kind of trace. Available kinds are defined in the <sys/subsys_id.h> header file as follows:
ERROR_TRACE_BIT Error tracing mask.
HDR_IN_BIT Inbound header tracing mask.
HDR_OUT_BIT Outbound header tracing mask.
LOGGING_TRACE_BIT Log call tracing mask.
LOOP_BACK_BIT For loopback.
PDU_IN_BIT Inbound PDU tracing mask.
PDU_OUT_BIT Outbound PDU tracing mask.
PROCEDURE_TRACE_BIT Procedure entry/exit trace.
PTOP_BIT For point to point.

DESCRIPTION
The KTRC_CK network service is a macro that verifies whether tracing is enabled for the current subsystem (given by subsys_id) and kind (given by trace_kind).

The difference between this macro and KTRC_CHECK macro is that KTRC_CK does not include the per-interface check.

RETURN VALUES
0 Tracing is disabled.
1 Tracing is enabled.

CONSTRAINTS
None

EXAMPLES
The enet driver uses this macro as follows to check if tracing is enabled for outbound data:
if (KTRC_CK(ENET_ID, PDU_OUT_BIT)
{
    ktrc_write(...);
}
SEE ALSO

ktrc_write(9F), KTRC_CHECK(9F)
NAME

ktrc_write - Sends trace messages to kernel trace and log facility.

SYNOPSIS

#include <sys/net_diag.h>
#include <sys/subsys_diag.h>
int ktrc_write(
    short subsys_id,
    u_signed kind,
    int path_id,
    int device_id,
    caddr_t tl_packet,
    int tl_packet_cnt
);

DESCRIPTION

The ktrc_write network function sends trace messages to the kernel trace and log facility. Prefiltering is done at the time of the trace call, and unwanted messages are dropped.

PARAMETERS

subsys_id   The unique subsystem ID of the calling subsystem (number assigned by Hewlett-Packard).

kind   The kind of trace. All kinds are defined in the header file <sys/subsys_id.h>. The following are the defined trace kind values. They can be ORed to produce the combination of trace kinds.

    ERROR_TRACE_BIT   Error tracing mask.
    HDR_IN_BIT       In bound header tracing mask.
    HDR_OUT_BIT      Outbound header tracing mask.
    LOGGING_TRACE_BIT Log call tracing mask.
    LOOP_BACK_BIT    For loopback.
    PDU_IN_BIT       Inbound PDU tracing mask.
    PDU_OUT_BIT      Outbound PDU tracing mask.
    PROCEDURE_TRACE_BIT Procedure entry/exit trace.
    PTOP_BIT         For point to point.
    STATE_TRACE_BIT  State machine tracing mask.

path_id   The connection path on the host. If this is a non-applicable parameter, pass in -1.

device_id   The device ID number (for example, enet driver’s instance number) of the calling subsystem message. If this is a nonapplicable parameter, pass in -1.

tl_packet   Either a pointer to an mblk chain or a pointer to a set of iovec structures as determined by tl_packet_cnt. The calling routine will pass a pointer (cast to caddr_t) to an mblk chain or an iovec structure. This structure is immediately copied into a memory location owned by tracing and logging facilities. Therefore, it is not necessary for the calling routine to copy the data and then pass a pointer to it.
tl_packet_cnt

If -2, tl_packet points to an mblk chain. If greater than 0, this is the number of the iovec structure to which tl_packet points.

RETURN VALUES
Always returns a 0.

CONSTRAINTS
None

SEE ALSO
KTRC_CHECK(9F), KTRC_CK(9F)
NAME
netmgr_arg_req_type_t - Specifies whether an option requires an argument or not

SYNOPSIS
#include <netmod.h>

DESCRIPTION
Indicates whether a given option can take arguments or not. The nc_arg_req field in netmgr_option_t is of the type netmgr_arg_req_type_t.

STRUCTURE MEMBERS
typedef enum netmgr_arg_req_type {
    NM_REQ = 1,
    NM_NOT_REQ,
    NM_OPTIONAL
} netmgr_arg_req_type_t;

NM_REQ = 1       Specifies that an argument is required for the option
NM_NOT_REQ       Specifies that an argument is not required for the option
NM_OPTIONAL     Specifies that an argument is optional

SEE ALSO
subsystem_netmgr(1M), netmgr_option_t(9S)
NAME

netmgr_check_class_in_dlpilist - Checks if a class is a valid class in the list of HP-DLPI registered interfaces.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_check_class_in_dlpilist(
    const char * class
);

PARAMETERS

class    Class that must be verified among the HP-DLPI registered interfaces.

DESCRIPTION

This routine checks whether the given class is a valid class among the HP-DLPI registered interfaces.

RETURN VALUES

0    Success. The class is a valid HP-DLPI registered one.

-1   Error. Class is not found among the HP-DLPI registered interfaces.

CONSTRAINTS

This API can be used by subsystems that are registered with HP-DLPI.

SEE ALSO

netmgr_check_class_inst_in_dlpilist(9F)
netmgr_check_class_inst_in_dlpilist(9F)

NAME

netmgr_check_class_inst_in_dlpilist - Checks if a class is a valid class in the list of HP-DLPI registered interfaces.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_check_class_inst_in_dlpilist(
    const char *class,
    int32_t inst
);

PARAMETERS

class Class of the interface that must be verified among the HP-DLPI registered interfaces.

inst Instance number of the interface must be verified among the HP-DLPI registered interfaces.

DESCRIPTION

This routine checks whether a given class instance is valid among the HP-DLPI registered interfaces.

RETURN VALUES

0 Success. The interface is a valid HP-DLPI registered one.

—1 Error. Interface cannot be found among the HP-DLPI registered interfaces.

CONSTRAINTS

This API can be used by subsystems that are registered with HP-DLPI.

SEE ALSO

netmgr_check_class_in_dlpilist (9F)
NAME

netmgr_clone_attrib - Creates a copy of an attribute.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_clone_attrib(
    const netmod_attrib_t *src_attrib,
    netmod_attrib_t **clone_attrib
);

PARAMETERS

src_attrib The source attribute.
clone_attrib The cloned attribute.

DESCRIPTION

This function can be used to create a duplicate copy of the source attribute.

RETURN VALUES

0 Success
Negative integer Error. Negative value of the appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

netmgr_free_cloned_attrib (9F)
netmgr_cmd_t(9S)

NAME

netmgr_cmd_t - Contains parsed command-line options and arguments

SYNOPSIS

#include <netmod.h>

DESCRIPTION

This structure contains the parsed command-line options and option arguments. The options are segregated into operations, targets, operational qualifiers, target qualifiers, and unknowns. Though attributes and subsystem qualifiers are target qualifiers, they are treated as separate option types to simply processing.

STRUCTURE MEMBERS

typedef struct {
    netmgr_nameval_t* nc_ops;
    netmgr_nameval_t* nc_targets;
    netmgr_nameval_t* nc_op_quals;
    netmgr_nameval_t* nc_tgt_quals;
    netmod_attrib_t* nc_bound_attribs;
    netmod_attrib_t* nc_unbound_attribs;
    netmgr_nameval_t* nc_subsys_quals;
    netmgr_nameval_t* nc_unknowns;
    uint32_t nc_num_ops;
    uint32_t nc_num_tgts;
    uint32_t nc_num_op_quals;
    uint32_t nc_num_tgt_quals;
    uint32_t nc_num_bound_attribs;
    uint32_t nc_num_unbound_attribs;
    uint32_t nc_num_subsys_quals;
    uint64_t nc_cmd_bitmask;
    uint64_t nc_flags;
    uint8_t nc_attrib_multi_letter;
    void* nc_rsvd
} netmgr_cmd_t;

nc_ops

Null terminated list of operations. As a rule, a maximum of two operations can be specified in the command-line with one of them being the --help operation.

nc_targets

Null terminated list of target options along with the arguments.

nc_op_quals

Null terminated list of operational qualifiers along with arguments, if any.

nc_tgt_quals

Null terminated list of target qualifiers along with arguments, if any.

nc_bound_attribs

Null terminated list of bound attributes along with values.

nc_unbound_attribs

Null terminated list of unbound attributes.

nc_subsys_quals

Null terminated list of subsystem qualifiers along with arguments, if any.

nc_unknowns

Null terminated list of options that are not recognized by the nwmgr parser, but might be understood by the subsystem.

nc_num_ops

Number of operations in the ops list (cannot be more than 2).
nc_num_tgts  Number of target options in the tgt list.
nc_num_op_quals  Number of operational qualifiers in the op_quals list.
nc_num_tgt_quals  Number of target qualifiers in the tgt_quals list.
nc_num_bound_attribbs  Number of attributes in the nc_bound_attribibs list.
nc_num_unbound_attribbs  Number of attributes in the nc_unbound_attribibs list.
nc_num_subsys_quals  Number of subsystem qualifiers in the subsys_quals list.
nc_num_unknowns  Number of unknown options in the nc_unknowns list.
nc_cmd_bitmask  Bitmask of all the options specified in command line.
nc_flags  Flag reserved for internal use.
nc_attrib_multi_letter  Flag to indicate whether an attribute option specified in command-line is in multi-letter or single letter format. If set to 0, indicates that the single-letter format was used or that the attribute option was not present in command line. If set to 1, indicates that multi-letter format was used. If multiple attribute options are specified, the flag indicates the format of the first occurrence of the option.
nc_rsvd  Reserved for future use.

SEE ALSO

subsystem_nwmgr(1M), netmgr_get_class_in_cmd (9F), netmgr_get_inst_in_cmd (9F), netmgr_get_ifname(9F), netmgr_get_dlpi_ifname(9F), netmgr_verify_one_instance(9F), netmgr_validate_cmd(9F)
NAME

netmgr_conf_delete_attrib - Deletes all entries in the configuration file that matches the attribute name.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_conf_delete_attrib(
    int32_t fd,
    netmod_attrib_t *nameval
);

PARAMETERS

fd File descriptor obtained through the netmgr_conf_get_fd service.
nameval The netmod_attrib_t structure that contains the name of the attribute to be deleted.

DESCRIPTION

This function deletes all entries that match the attribute name given in nameval from the configuration file.

RETURN VALUES

0 Success
Negative integer Error. Negative value of the appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

netmgr_conf_delete_nameval_for_idx(9F), netmgr_conf_delete_idx(9F)
netmgr_conf_delete_idx (9F)

NAME

netmgr_conf_delete_idx - Deletes all entries from the configuration file that matches the index.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_conf_delete_idx(
    int32_t fd,
    int32_t index
);

PARAMETERS

fd File descriptor obtained through the netmgr_conf_get_fd service.
index Index of the attribute entries that will be deleted from the configuration file.

DESCRIPTION

This function deletes all entries from the configuration file that matches the index given as input parameter.

RETURN VALUES

0 Success

Negative integer Error. Negative value of appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

netmgr_conf_delete_nameval_for_idx(9F), netmgr_conf_delete_attrib(9F)
NAME

netmgr_conf_delete_nameval_for_idx() - Deletes an entry from the configuration file, based on attribute name and index value passed as input parameter.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_conf_delete_nameval_for_idx(
    int32_t fd,
    netmod_attrib_t *nameval,
    int32_t index
);

PARAMETERS

fd File descriptor obtained through the netmgr_conf_get_fd service.
nameval The netmod_attrib_t structure that contains the name of the attribute.
index Index of the attribute entry that is deleted from the configuration file.

DESCRIPTION

This function deletes the entry from the configuration file that matches with the attribute name and index value given in nameval structure and index input parameters, respectively.

RETURN VALUES

0 Success
Negative integer Error. Negative value of appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

netmgr_conf_delete_idx(9F), netmgr_conf_delete_attrib(9F)
**NAME**

`netmgr_conf_get_atlist_for_idx` - Retrieves the list of attributes for a given index.

**SYNOPSIS**

```c
#include <netmod.h>

int32_t (d, int32_t index, , void(**free_attrib_list(netmod_attrib_t
*attrib_list))); netmgr_conf_get_atlist_for_idx(
  int32_t fd,
  int32_t index,
  netmod_attrib_t **attrib_list,
  void (**free_attrib_list(netmod_attrib_t *attrib_list))
);
```

**PARAMETERS**

- **fd**
  File descriptor obtained through the `netmgr_conf_get_fd` service.

- **index**
  Index for which the name and value of the attribute is retrieved from the configuration file.

- **attrib_list**
  This contains the list of `netmod_attrib_t` structures for every attribute retrieved for the given index.

- **free_attrib**
  Function pointer to the `nwmgr` service routine that frees the memory allocated for `attrib_list`. Must be invoked by the caller at the appropriate time while processing `attrib_list`.

**DESCRIPTION**

This function retrieves the list of attributes for a given index. It sets the name and value to `na_name` and `na_value` field of `netmod_attrib_t` structure of each attribute in the list.

**RETURN VALUES**

- 0 or positive integer: Success. Number of attributes returned in `attrib_list`.
- Negative integer: Error. Negative value of appropriate error code as defined in `<sys/errno.h>`.

**CONSTRAINTS**

None

**SEE ALSO**

`netmgr_conf_get_fd(9F)`, `netmgr_conf_get_indices_for_nameval(9F)`, `netmgr_conf_get_indices_for_name(9F)`, `netmgr_conf_get_idx_list(9F)`, `netmgr_conf_get_val_for_name_idx(9F)`
NAME

netmgr_conf_get_fd - Opens the configuration file.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_conf_get_fd(
    char *conf_file_name
);

PARAMETERS

conf_file_name Name of the configuration file to be opened.

DESCRIPTION

This function opens the configuration file (in read and write mode) specified as input parameter.

RETURN VALUES

Positive integer Success. File descriptor of the newly opened configuration file
Negative integer Error. Negative value of appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

netmgr_conf_get_indices_for_nameval(9F), netmgr_conf_get_indices_for_name(9F),
netmgr_conf_get_indices_for_name_idx(9F), netmgr_conf_get_idx_list(9F),
netmgr_conf_get_val_for_name_idx(9F), netmgr_conf_get_atlist_for_idx(9F)
netmgr_conf_get_idx_list(9F)

NAME

netmgr_conf_get_idx_list - Retrieves the list of unique indices present in the configuration file.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_conf_get_idx_list(
    int32_t fd,
    int32_t* index_list[],
    int32_t max_inds
);

PARAMETERS

fd File descriptor obtained through the netmgr_conf_get_fd service
index_list List of unique indices in the configuration file.
max_inds Maximum number of indices that the pre-allocated array, index_list, can hold.

DESCRIPTION

This function retrieves the list of unique indices present in the configuration file. If any global entries are present (entries without indices), they are represented by a single entry with -1 in the index list. Multiple entries with duplicate indices are not reported, and are represented by a single entry in the index list. The array index_list must be pre-allocated by the caller and the maximum size of the pre-allocated array must be passed as input parameter in max_inds. If the number of unique indices exceeds max_inds, the function is restricted by max_inds.

RETURN VALUES

0 or positive integer Success. Number of unique indices returned in index_list.
Negative integer Error. Negative value of appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

netmgr_conf_get_fd(9F), netmgr_conf_get_indices_for_nameval (9F),
netmgr_conf_get_indices_for_name(9F), netmgr_conf_get_atlist_for_idx(9F), netmgr_conf_get_val_for_name_idx(9F)
**NAME**

netmgr_conf_get_indices_for_name - Retrieves indices in sorted order for entries in the configuration file that matches the name of the attribute.

**SYNOPSIS**

```
#include <netmod.h>

int32_t (**free_index_list(int32_t *index_list)));
netmgr_conf_get_indices_for_name(
    int32_t fd,
    const netmod_attrib_t *attr,
    int32_t **index_list,
    void (**free_index_list(int32_t *index_list)));
```

**PARAMETERS**

- **fd** File descriptor obtained through the `netmgr_conf_get_fd` service.
- **attr** The `netmod_attrib_t` structure that contains the name of the attribute to be retrieved from the configuration file.
- **index_list** This contains the list of indices.
- **free_index_list** Function pointer to the `nwmgr` service routine that frees the memory allocated for `index_list`. Must be invoked by the caller at the appropriate time while processing `index_list`.

**DESCRIPTION**

This function retrieves indices in sorted order for entries in the configuration file, which match the name of the attribute specified as input parameter.

**RETURN VALUES**

- **0 or Positive integer** Success. Number of indices returned in `index_list`.
- **Negative integer** Error. Negative value of appropriate error code as defined in `<sys/errno.h>`.

**CONSTRAINTS**

None

**SEE ALSO**

- `netmgr_conf_get_fd (9F)`, `netmgr_conf_get_indices_for_nameval(9F)`,
- `netmgr_conf_get_indices_for_name_idx(9F)`, `netmgr_conf_get_idx_list(9F)`,
- `netmgr_conf_get_val_for_name_idx(9F)`, `netmgr_conf_get_atlist_for_idx(9F)`
netmgr_conf_get_indices_for_nameval(9F)

NAME

netmgr_conf_get_indices_for_nameval - Retrieves indices in sorted order for entries in the configuration file.

SYNOPSIS

#include <netmod.h>

int32_t (int32_t fd, nameval, int32_t **index_list,
    void(**free_index_list(int32_)); netmgr_conf_get_indices_for_nameval(
    int32_t fd,
    const netmod_attrib_t *nameval,
    int32_t **index_list,
    void(**free_index_list(int32_t *index_list))
    );

PARAMETERS

fd File descriptor obtained through the netmgr_conf_get_fd service.

nameval The netmod_attrib_t structure that contains the name and value of the attribute to be retrieved from the configuration file.

index_list This contains the list of indices.

free_index_list Function pointer to the nwmgr service routine that frees the memory allocated for index_list. Must be invoked by the caller at the appropriate time while processing index_list.

DESCRIPTION

This function retrieves indices in sorted order for entries in the configuration file, which match the name and value of the attribute specified as input parameter.

RETURN VALUES

0 or positive integer Success. Number of indices returned in index_list.

Negative integer Error. Negative value of appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

netmgr_conf_get_fd(9F), netmgr_conf_get_indices_for_name(9F),
netmgr_conf_get_indices_for_name_idx(9F), netmgr_conf_get_idx_list(9F),
netmgr_conf_get_val_for_name_idx(9F), netmgr_conf_get_atlist_for_idx(9F)
netmgr_conf_get_val_for_name_idx(9F)

NAME

netmgr_conf_get_val_for_name_idx - Retrieves value of the input attribute.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_conf_get_val_for_name_idx(
    int32_t fd,
    netmod_attrib_t *attr,
    int32_t index
);

PARAMETERS

fd File descriptor obtained through the netmgr_conf_get_fd service.
attr The netmod_attrib_t structure that contains name of the input attribute.
index Index of the attribute entry in the configuration file.

DESCRIPTION

This function retrieves value of the input attribute for the given index from the configuration file. It sets the retrieved value to na_value field of the input netmod_attrib_t structure.

RETURN VALUES

0 Success
Negative integer Error. Negative value of appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

netmgr_conf_get_fd(9F), netmgr_conf_get_indices_for_nameval(9F), netmgr_conf_get_indices_for_name(9F), netmgr_conf_get_idx_list(9F), netmgr_conf_get_atlist_for_idx(9F)
NAME

netmgr_conf_set_atlist_for_idx - Sets the list of attributes for a given index

SYNOPSIS

#include <netmod.h>

int32_t netmgr_conf_set_atlist_for_idx(
    int32_t fd,
    int32_t index,
    netmod_attrib_t *attrib_list
);

PARAMETERS

fd File descriptor obtained through the netmgr_conf_get_fd service.
index Index for which the name and value of the attribute is retrieved from the configuration file.
attrib_list This contains the list of netmod_attrib_t structures for every attribute retrieved for the given index.

DESCRIPTION

This function sets the list of attributes for a given index. It sets the name and value to the na_name and na_value fields of netmod_attrib_t structure of each attribute in the list. If the index does not exist, this routine creates a new entry for the index in the configuration file and inserts the list of attributes for it.

RETURN VALUES

0 Success
Negative integer Error. Negative value of appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

subsystem_nwmgr(1M)
NAME

netmgr_conf_set_attrib_value - Sets or modifies the attribute value

SYNOPSIS

#include <netmod.h>

int32_t netmgr_conf_set_attrib_value(
    int32_t fd,
    netmod_attrib_t *nameval,
    int32_t index
);

PARAMETERS

fd File descriptor obtained through the netmgr_conf_get_fd service.
nameval The netmod_attrib_t structure that contains the name of the attribute along with its value, which is set in the configuration file.
index Index of the attribute for which the value is set.

DESCRIPTION

This function sets or modifies the attribute value for the entry corresponding to a given attribute name and index to na_value in nameval structure. If the attribute does not exist for the particular index, a new entry for the attribute is created.

RETURN VALUES

0 Success
Negative integer Error. Negative value of the appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

subsystem_nwmgr(1M)
NAME
netmgr_dlpi_attach - Attaches the DLPI stream to the specified instance

SYNOPSIS
#include <sys/dlpi.h>

int32_t netmgr_dlpi_attach(
    int32_t fd,
    int32_t instance
);

PARAMETERS
fd File descriptor opened on the /dev/dlpi device file.
instance Interface to which the stream, represented by file descriptor fd, must be attached.

DESCRIPTION
This function attaches the stream, opened through netmgr_dlpi_open, to the instance specified as input parameter.

RETURN VALUES
0 Success
EIO Failure

CONSTRAINTS
This API can be used by LAN subsystems that are tightly coupled to HP-DLPI.

SEE ALSO
netmgr_dlpi_open (9F), netmgr_dlpi_detach(9F)
netmgr_dlpi_detach(9F)

NAME

netmgr_dlpi_detach - Detaches the stream from its currently attached interface

SYNOPSIS

# include <sys/dlpi.h>

int32_t netmgr_dlpi_detach(
    int32_t fd
);

PARAMETERS

fd       File descriptor opened on the /dev/dlpi device file and attached to an interface.

DESCRIPTION

This function detaches the stream represented by fd from its currently attached interface.

RETURN VALUES

0       Success
EIO     Failure

CONSTRAINTS

This API can be used by LAN subsystems that are tightly coupled to HP-DLPI.

SEE ALSO

netmgr_dlpi_open(9F), netmgr_dlpi_attach(9F)
netmgr_dlpi_open(9F)

NAME

netmgr_dlpi_open - Opens the /dev/dlpi device file and returns the file descriptor

SYNOPSIS

#include <sys/dlpi.h>

int32_t netmgr_dlpi_open(void);

PARAMETERS

None

DESCRIPTION

This function opens the device file /dev/dlpi and returns the file descriptor to the caller.

RETURN VALUES

Positive integer Success. File descriptor obtained by opening /dev/dlpi.
EACCES Failure. Unable to open the device file.

CONSTRAINTS

This API can be used by LAN subsystems that are tightly coupled to HP-DLPI.

SEE ALSO

netmgr_dlpi_attach(9F), netmgr_dlpi_detach(9F)
netmgr_find_num_opts(9F)

NAME
netmgr_find_num_opts - Finds the number of occurrences of an option based on its multi-letter name

SYNOPSIS
#include <netmod.h>

int32_t netmgr_find_num_opts(
     const netmgr_nameval_t *optlist,
     const char *optname
);

PARAMETERS
optlist List of the netmgr_nameval_t options.
optname Name of the option that must be found in optlist.

DESCRIPTION
Find the number of occurrences of an option within a given list based on its multi-letter name.

RETURN VALUES
0 or Positive integer Success. Integer count of the number of occurrences of the option.
-1 Error

CONSTRAINTS
None

SEE ALSO
netmgr_find_opt(9F), netmgr_find_opt_by_value(9F), subsystem_nwmgr (1M)
NAME

netmgr_find_opt - Searches for a particular option based on a multi-letter name

SYNOPSIS

#include <netmod.h>

netmgr_nameval_t *netmgr_find_opt(
    netmgr_nameval_t optlist,
    char *optname
);

PARAMETERS

optlist List of the netmgr_nameval_t structures.
optname The multi-letter form of the option to be searched in optlist.

DESCRIPTION

This routine searches for a particular option within a list of netmgr_nameval_t structures, if the multi-letter name is specified.

RETURN VALUES

Non-NULL value Success. Pointer to the netmgr_nameval_t structure for the option.
NULL Failure

CONSTRAINTS

None

SEE ALSO

netmgr_find_opt_for_int(9F), netmgr_find_opt_by_value(9F), netmgr_find_num_opts(9F), subsystem_nwmgr(1M)
**NAME**

netmgr_find_opt_by_value - Searches a particular option based on its name and value

**SYNOPSIS**

#include <netmod.h>

netmgr_nameval_t *netmgr_find_opt_by_value(
    netmgr_nameval_t optlist,
    const char *opt_name,
    const char *opt_value
);

**PARAMETERS**

- **optlist** List of the netmgr_nameval_t structures.
- **opt_name** Option name to be searched in optlist.
- **opt_value** Option value to be searched in optlist.

**DESCRIPTION**

This routine searches for a particular option within a list of netmgr_nameval_t structures, if the name and value are specified.

**RETURN VALUES**

- Non-NULL value Success. Pointer to the netmgr_nameval_t structure for the option.
- NULL Failure

**CONSTRAINTS**

None

**SEE ALSO**

netmgr_find_opt_for_int (9F), netmgr_find_num_opts (9F), netmgr_find_opt(9F), subsystem_nwmgr(1M)
netmgr_find_opt_for_int(9F)

NAME
netmgr_find_opt_for_int - Searches a particular option based on its integer equivalent

SYNOPSIS
#include <netmod.h>
netmgr_nameval_t *netmgr_find_opt_for_int(
    netmgr_nameval_t optlist,
    const uint64_t opt_int
);

PARAMETERS
optlist List of the netmgr_nameval_t structures.
opt_int Integer equivalent of the option to be searched in optlist.

DESCRIPTION
This routine searches for a particular option within a list of netmgr_nameval_t structures, if the equivalent integer is specified.

RETURN VALUES
Non-NULL value Success. Pointer to the netmgr_nameval_t structure for the option.
NULL Failure

CONSTRAINTS
None

SEE ALSO
netmgr_find_opt_by_value(9F), netmgr_find_num_opts(9F), netmgr_find_opt(9F), subsystem_nwmgmtr(1M)
NAME
netmgr_free_cloned_attrib - Frees the cloned attribute

SYNOPSIS
#include <netmod.h>
void netmgr_free_cloned_attrib(
    netmod_attrib_t *attrib
);

PARAMETERS
attrib    The attribute to be freed.

DESCRIPTION
This function frees the cloned attribute.

RETURN VALUES
None

CONSTRAINTS
None

SEE ALSO
netmgr_clone_attrib(9F)
netmgr_get_class_in_cmd(9F)

NAME

netmgr_get_class_in_cmd - Extracts the class name from the command-line options

SYNOPSIS

#include <netmod.h>

int32_t netmgr_get_class_in_cmd(
    const netmgr_cmd_t *cmd,
    char *class_name
);

PARAMETERS

cmd          Command structure.
class_name   Class name specified in the command, either in the form of --class option or as part of --class_instance. The caller must preallocate memory for the string.

DESCRIPTION

The class name is extracted from the options specified on the command line.

RETURN VALUES

cmd          Command structure.
class_name   Class name specified in the command, either in the form of --class option or as part of --class_instance. The caller must preallocate memory for the string.

CONSTRAINTS

None

SEE ALSO

netmgr_get_instance (9F), netmgr_get_inst_in_cmd (9F), netmgr_get_ifname(9F), netmgr_get_dlpi_ifname(9F), subsystem_nwmgr(1M)
NAME

netmgr_get_dlpi_fd - Opens and attaches device file for the LAN interface

SYNOPSIS

#include <netmod.h>

int32_t netmgr_get_dlpi_fd(
    const int32_t instance
);

PARAMETERS

instance Instance number of the interface. If the user specified class_instance on the command-line, it must be converted to an instance.

DESCRIPTION

This function opens the device file for the specified LAN interface and attaches to its PPA. The dl_device_name field in dl_hp_ext_ppa_info_t structure for the interface is used to form the device filename.

RETURN VALUES

Positive integer Success. File descriptor of the opened and attached interface.
EINVAL Failure. Invalid PPA.
EACCES Failure. Unable to open the device file.
EIO Failure. Attach to the PPA failed.
ENOMEM Failure. Memory allocation failed.

CONSTRAINTS

None

SEE ALSO

netmgr_dlpi_open(9F), netmgr_get_ppa_list(9F), get_subsys_ppa_list(9F), netmgr_get_ppa_info(9F)
NAME

netmgr_get_dlpi_ifname - Checks if the class_instance contains only numerals and HP-DLPI registered interfaces

SYNOPSIS

#include <netmod.h>

int32_t netmgr_get_dlpi_ifname(
    const netmgr_cmd_t *cmd,
    char *ifname
);

PARAMETERS

cmd Command structure.
ifname Class instance formed from the options specified in the command-line and validated as a valid HP-DLPI registered interface. The caller must preallocate memory for this variable.

DESCRIPTION

If options in the command-line contain --class_instance or --class --instance, this routine forms the class_instance and validates that the instance contains only numerals. In addition, it validates the class_instance among the HP-DLPI registered interfaces.

If --subsystem is specified along with --instance, this routine gets the class from HP-DLPI and forms the class_instance. This service must be used only on existing interfaces for operations, such as --get, --set, --reset, --enable, --disable, --cra, --delete, and --diagnose. If the nwmgr command-line creates a new virtual interface using --add operation, this service returns EINVAL because the interface is not HP-DLPI registered.

RETURN VALUES

0 Success
EINVAL Error. Badly formed class_instance, or class_instance not found among HP-DLPI reported list of interfaces.

CONSTRAINTS

None

SEE ALSO

netmgr_get_instance (9F), netmgr_get_class_in_cmd (9F), netmgr_get_inst_in_cmd (9F), netmgr_get_ifname (9F), subsystem_nwmgr (1M)
NAME
netmgr_get_dlpi_stats - Gets the internal statistics of the file descriptor

SYNOPSIS
#include <sys/dlpi_ext.h>

int32_t netmgr_get_dlpi_stats(
    int32_t fd,
    dl_hp_internal_stats_t *int_stats
);

PARAMETERS
fd File descriptor of the stream that is opened and attached to an interface.
int_mibstats The internal statistics for the currently attached interface.

DESCRIPTION
This function gets the internal statistics of the file descriptor that has been opened and attached
to an interface, by invoking the DL_HP_GET_INTERNAL_STATS_REQ primitive. The caller must
preallocate memory for storing the dl_hp_internal_stats_t structure.

RETURN VALUES
0 Success
EIO Failure. HP-DLPI primitive invocation failed.

CONSTRAINTS
This API can be used by subsystems that are registered with HP-DLPI.

SEE ALSO
netmgr_get_dlpi_usage_info (9F), netmgr_get_macaddr (9F), netmgr_get_fact_macaddr (9F),
netmgr_get_iface_type(9F), netmgr_get_mibstats (9F)
netmgr_get_dlpi_usage_info(9F)

NAME

netmgr_get_dlpi_usage_info - Gets the usage information of the file descriptor.

SYNOPSIS

#include <sys/dlpi_ext.h>

int32_t netmgr_get_dlpi_usage_info(
    int32_t fd,
    dl_hp_usage_info_t **usage_info,
    void (**free_func)(dl_hp_usage_info_t *)
);

PARAMETERS

fd File descriptor that was successfully returned in a previous
     netmgr_get_dlpi_fd call.
usage_info The usage info for the currently attached interface.
free_func Pointer to the function to be invoked for freeing memory allocated for usage_info.

DESCRIPTION

This function gets the usage information of the file descriptor that is opened and attached to an
interface in the form of the dl_hp_usage_info_t structure. The size of the usage_info variable
varies depending on the number of upper layer modules and applications that are attached to
it. Memory for the variable is allocated by the routine. The caller must invoke the free_func
function to free the allocated memory after the process is completed.

RETURN VALUES

0 Success
EIO Failure. HP-DLPI primitive invocation failed.
ENOMEM Failure. Memory allocation failed.

CONSTRAINTS

Subsystems that are registered with DLPI.

SEE ALSO

netmgr_get_macaddr(9F), netmgr_get_fact_macaddr (9F), netmgr_get_iface_type(9F),
netmgr_get_mibstats(9F), netmgr_get_64bit_stats(9F), netmgr_get_dlpi_stats(9F)
netmgr_get_fact_macaddr(9F)

NAME

netmgr_get_fact_macaddr - Gets the factory MAC address and its length of an interface.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_get_fact_macaddr(
    int32_t fd,
    u_char *macaddr,
    int32_t *macaddr_len
);

PARAMETERS

fd File descriptor that was successfully returned in a previous netmgr_get_dlpi_fd call.
macaddr The factory MAC address of the interface specified by fd.
macaddr_len Length of the factory MAC address of the interface specified by fd.

DESCRIPTION

This function gets the factory MAC address and its length. The interface is specified by the file descriptor (fd) which can be obtained by calling netmgr_get_dlpi_fd. The caller must pass a valid memory for storing the factory MAC address and the MAC address length.

RETURN VALUES

Positive integer Success. Valid factory MAC address and length of the MAC address.
EIO Failure. HP-DLPI primitive invocation failed.

CONSTRAINTS

This API cannot be used by subsystems that are not registered with HP-DLPI.

SEE ALSO

netmgr_get_dlpi_usage_info(9F) , netmgr_get_macaddr(9F) , netmgr_get_iface_type (9F) ,
netmgr_get_mibstats(9F) , netmgr_get_64bit_stats (9F) , netmgr_get_dlpi_stats (9F)
netmgr_get_iface_type (9F)

NAME

netmgr_get_iface_type - Gets the string that defines the interface type of the file descriptor

SYNOPSIS

#include <netmod.h>

int32_t netmgr_get_iface_type(int32_t fd, char *iface_type);

PARAMETERS

fd File descriptor that was successfully returned in a previous netmgr_get_dlpi_fd call.
iface_type The interface type string for the currently attached interface.

DESCRIPTION

This function gets the string that defines the interface type of the file descriptor that has been opened and attached to an interface by invoking the DL_HP_GET_DRV_IOC ioctl. The caller must preallocate memory for storing the interface type string.

RETURN VALUES

0 Success
EIO Failure. HP-DLPI ioctl invocation failed.

CONSTRAINTS

This API can be used by subsystems that are registered with HP-DLPI.

SEE ALSO

netmgr_get_dlpi_usage_info (9F), netmgr_get_macaddr (9F), netmgr_get_fact_macaddr (9F), netmgr_get_mibstats(9F), netmgr_get_64bit_stats (9F), netmgr_get_dlpi_stats (9F)
NAME

netmgr_get_ifname - Checks if the class_instance contains only numerals

SYNOPSIS

#include <netmod.h>

int32_t netmgr_get_ifname(
    const netmgr_cmd_t *cmd,
    char *ifname
);

PARAMETERS

cmd Command structure.
ifname Class instance formed from the options specified on the command line. The caller must preallocate memory for this variable.

DESCRIPTION

If options on the command line contain either --class_instance or --class --instance, this routine forms the class_instance and validates that the instance contain only numerals.

RETURN VALUES

0 Success
EINVAL Error. Badly formed class_instance.

CONSTRAINTS

None

SEE ALSO

netmgr_get_instance(9F), netmgr_get_class_in_cmd(9F), netmgr_get_inst_in_cmd(9F), netmgr_get_dlpi_ifname(9F), subsystem_nwmgr (1M)
netmgr_get_inst_in_cmd(9F)

NAME

netmgr_get_inst_in_cmd - Extracts the instance from the command-line options

SYNOPSIS

#include <netmod.h>

int32_t netmgr_get_inst_in_cmd(
    const netmgr_cmd_t *cmd,
    int32_t *inst
); 

PARAMETERS

cmd Command structure.

inst Instance number specified on the command line, either as --instance or as part of --class_instance.

DESCRIPTION

This function extracts the instance specified on the command line.

RETURN VALUES

0 Success

EINVAL Error. Badly formed instance, or instance not found on the command line.

CONSTRAINTS

None

SEE ALSO

netmgr_get_instance(9F), netmgr_get_class_in_cmd(9F), netmgr_get_ifname(9F), netmgr_get_dlpi_ifname(9F), subsystem_nwmgr(IM)
NAME

netmgr_get_instance - Extracts the instance number of a class instance

SYNOPSIS

#include <netmod.h>

int32_t netmgr_get_instance(
    char *cl_inst
);

PARAMETERS

cl_inst String representing class instance.

DESCRIPTION

For a given class instance in the string form, this routine extracts the instance number.

RETURN VALUES

Positive integer Success. Instance number.
EINVAL Error. Badly formed class_instance.

CONSTRAINTS

None

SEE ALSO

netmgr_get_class_in_cmd (9F), netmgr_get_inst_in_cmd(9F), netmgr_get_ifname (9F),
netmgr_get_dlpi_ifname(9F), subsystem_nwmgr (9S)
NAME

netmgr_get_macaddr - Provides the MAC address and the MAC address length of the file descriptor

SYNOPSIS

#include <netmod.h>

int32_t netmgr_get_macaddr(
    int32_t fd,
    u_char *const macaddr
);

PARAMETERS

fd File descriptor that was successfully returned in a previous netmgr_get_dlpi_fd call.

macaddr The MAC address of the currently attached interface.

DESCRIPTION

This function gets the MAC address and MAC address length of the file descriptor that is opened and attached to an interface from HP-DLPI. The caller must preallocate memory for storing the MAC address.

RETURN VALUES

Positive integer Success. Length of MAC address in macaddr.
EIO Failure. HP-DLPI primitive invocation failed.

CONSTRAINTS

This API can be used by subsystems that are registered with HP-DLPI.

SEE ALSO

netmgr_get_dlpi_usage_info (9F), netmgr_get_fact_macaddr (9F), netmgr_get_iface_type (9F), netmgr_get_mibstats (9F), netmgr_get_64bit_stats (9F), netmgr_get_dlpi_stats (9F)
NAME
netmgr_get_mibstats - Gets the MIB statistics of the file descriptor

SYNOPSIS
#include <sys/mib.h>
#include <sys/dlpi.h>

int32_t netmgr_get_mibstats(
    int32_t fd,
    Ext_mib_t *mib_stats
);

PARAMETERS
fd descriptor of the stream that is opened and attached to an interface.
mib_stats The MIB statistics for the currently attached interface.

DESCRIPTION
This function gets the MIB statistics of the file descriptor that is opened and attached to an interface by invoking the DL_GET_STATISTICS primitive. The caller must preallocate memory for storing the Ext_mib_t structure.

RETURN VALUES
0 Success
EIO Failure. HP-DLPI primitive invocation failed.

CONSTRAINTS
This API can be used by subsystems that are registered with HP-DLPI.

SEE ALSO
netmgr_get_dlpi_usage_info (9F), netmgr_get_macaddr (9F), netmgr_get_fact_macaddr (9F), netmgr_get_iface_type(9F), netmgr_get_64bit_stats (9F), netmgr_get_dlpi_stats (9F)
netmgr_get_ppa_info(9F)

NAME

netmgr_get_ppa_info - Obtains the dl_hp_ext_ppa_info_t structure

SYNOPSIS

#include <sys/dlpi.h>
#include <sys/dlpi_ext.h>
#include <netmod.h>

int32_t netmgr_get_ppa_info(
    const int32_t instance,
    dl_hp_ext_ppa_info_t *ppa_info
);

PARAMETERS

instance Instance number of the interface. If the user specifies a class_instance, it must be converted to instance.

ppa_info The dl_hp_ext_ppa_info_t structure containing information related to interface.

DESCRIPTION

If the interface name is provided in the form of lan_instance_number, this function will obtain the dl_hp_ext_ppa_info_t structure from HP-DLPI.

CONSTRAINTS

This API can be used by subsystems that are registered with HP-DLPI.

SEE ALSO

netmgr_get_dlpi_fd(9F), netmgr_dlpi_open(9F), netmgr_get_ppa_list(9F), netmgr_get_subsys_ppa_list(9F)
**NAME**

netmgr_get_ppa_list - Obtains the list of structures associated with all the interfaces found.

**SYNOPSIS**

```c
#include <sys/dlpi.h>
#include <sys/dlpi_ext.h>
#include <netmod.h>

int32_t ( ppa_list, )
 netmgr_get_ppa_list(
    dl_hp_ext_ppa_info_t **const ppa_list,
    void (**free_ppa_list)(dl_hp_ext_ppa_info_t *ppa_list))
);
```

**PARAMETERS**

- **ppa_list**
  - Contains the list of `dl_hp_ext_ppa_info_t` structures, one per interface.
  - If no interfaces are found or an error is encountered, this parameter contains NULL.

- **free_ppa_list**
  - Function pointer to the `nwmgr` service routine that frees the memory allocated for `ppa_list`. This field must be invoked by the caller at the appropriate time while processing `ppa_list`.

**DESCRIPTION**

This function obtains the list of `dl_hp_ext_ppa_info_t` structures from HP-DLPI by invoking the `dl_hp_ext_ppa_req` primitive. The caller must free the memory by invoking the `*free_ppa_list` function because the structure dynamically allocates the memory required to hold the list of structures.

**RETURN VALUES**

- 0 or positive integer: Success. The number of `dl_hp_ext_ppa_info_t` structures returned in `ppa_list`.
- EIO: Failure. HP-DLPI primitive invocation failed.

**CONSTRAINTS**

This API can be used by subsystems that are registered with HP-DLPI.

**SEE ALSO**

- `netmgr_dlpi_open(9F)`
- `netmgr_get_dlpi_fd(9F)`
- `netmgr_get_subsys_ppa_list(9F)`
- `netmgr_get_ppa_info(9F)`

*734 Network Device Driver Reference Pages*
NAME
netmgr_get_subsys_ppa_list - Obtains list of structures assigned for a subsystem

SYNOPSIS
# include <sys/dlpi.h>
# include <sys/dlpi_ext.h>
#include <netmod.h>

int32_t get_subsys_ppa_list(
    char *subsys,
    dl_hp_ext_ppa_info_t **const ppa_list,
    void (**)free_ppa_list(dl_hp_ext_ppa_info_t *ppa_list))
);

PARAMETERS
subsys Name of the subsystem, ppa_list, for which the information is to be obtained.
ppa_list Contains the list of dl_hp_ext_ppa_info_t structures, one per interface. If no interfaces are found or an error is encountered, this parameter contains NULL.
free_ppa_list Function pointer to the nwmgr service routine that frees the memory allocated for ppa_list. It must be invoked by the caller at the appropriate time while processing ppa_list.

DESCRIPTION
This function obtains the list of dl_hp_ext_ppa_info_t structures from HP-DLPI for a given subsystem by invoking the dl_hp_ext_ppa_req primitive. The caller must free the memory by invoking free_ppa_list function, because the structure dynamically allocates the memory required to hold the list of structures.

RETURN VALUES
0 or positive integer Success. The number of dl_hp_ext_ppa_info_t structures returned in ppa_list.
ENOMEM Failure. Memory allocation failed.
EIO Failure. HP-DLPI primitive invocation failed.

CONSTRAINTS
This API can be used by subsystems that are registered with HP-DLPI.

SEE ALSO
netmgr_dlpi_open(9F), netmgr_get_ppa_list(9F), netmgr_get_ppa_info(9F), netmgr_get_dlpi_fd(9F)
**NAME**

netmgr_get_64bit_stats - Gets the 64-bit MIB statistics of the file descriptor

**SYNOPSIS**

```c
#include <sys/mib.h>
#include <sys/dlpi.h>

int32_t netmgr_get_64bit_stats(
    int32_t fd,
    ext_64bit_mib_t *ext_mibstats
);
```

**PARAMETERS**

- `fd` - File descriptor of the stream that is opened and attached to an interface.
- `ext_mibstats` - The extended MIB statistics for the currently attached interface.

**DESCRIPTION**

This function gets the 64-bit MIB statistics of the file descriptor that has been opened and attached to an interface by invoking the `DL_GET_64BIT_STATS_REQ` primitive. The caller must preallocate memory for storing the `ext_64bit_mib_t` structure.

**RETURN VALUES**

- `0` Success
- `-EIO` Failure. HP-DLPI primitive invocation failed.

**CONSTRAINTS**

This API can be used by subsystems that are registered with HP-DLPI

**SEE ALSO**

- `netmgr_get_dlpi_usage_info (9F)`, `netmgr_get_macaddr(9F)`, `netmgr_get_fact_macaddr(9F)`, `netmgr_get_iface_type (9F)`, `netmgr_get_mibstats (9F)`, `netmgr_get_dlpi_stats (9F)`
name

netmgr_linkloop - Performs linkloop diagnostics

Synopsis

#include <netmod.h>

int32_t netmgr_linkloop(
    int32_t fd,
    netmgr_linkloop_t *linkloop_info,
    netmgr_linkloop_stats_t *linkloop_stats
);

Parameters

fd File descriptor obtained through the netmgr_get_dlpi_fd service routine.

linkloop_info Structure containing all the necessary information for linkloop diagnostics to be performed. If the command line options contain combinations of --time, --interval, and --iterations. The caller can convert these into intervals and iterations. The conversion also includes assignment of default values for the three time related options.

linkloop_stats Structure containing the number of test packets sent and received. The caller can deduce the percentage of packets dropped from these two, if required.

Description

This function performs linkloop diagnostics from a specific interface to the MAC address of the destination interface. The MAC address must be specified as an attribute.

Return Values

0 Success

Negative integer Error. Negative value of the appropriate error code as defined in <sys/errno.h>.

Constraints

None

See Also

netmgr_show_linkloop_stats(9F)
NAME

netmgr_linkloop_stats_t - Contains statistics of the linkloop diagnostics

SYNOPSIS

#include <netmod.h>

DESCRIPTION

This structure holds the statistics of linkloop diagnostics. It contains the number of test packets sent and number of test packets received.

STRUCTURE MEMBERS

typedef struct {
  int64_t nt_sent;
  int64_t nt_recv_ok;
  int64_t nt_recv_bad_data;
  int64_t nt_recv_bad_len;
  int64_t nt_recv_bad_header;
  int64_t nt_recv_timeout
}netmgr_linkloop_stats_t;

nt_sent Number of test packets sent.
nt_recv_ok Number of test packets received without errors.
nt_recv_bad_data Number of test packets received with bad data.
nt_recv_bad_len Number of test packets received with bad packet size.
nt_recv_bad_header Number of test packets received with bad header.
nt_recv_timeout Number of missed test packets because of timeout.

SEE ALSO

netmgr_linkloop(9F), netmgr_show_linkloop_stats (9F)
netmgr_linkloop_t(9S)

NAME

netmgr_linkloop_t - Contains information about linkloop diagnostics

SYNOPSIS

#include <netmod.h>

DESCRIPTION

This structure contains information required to perform linkloop diagnostics.

STRUCTURE MEMBERS

typedef struct {
    uint32_t nt_pktsize;
    uint32_t nt_interval;
    uint32_t nt_iterations;
    uint32_t nt_timeout;
    uint8_t* nt_dest_macaddr;
    uint8_t* nt_src_macaddr;
    int32_t nt_macaddr_len
} netmgr_linkloop_t;

nt_pktsize Size of the test packet in bytes.
nt_interval Time interval between successive test packets.
nt_iterations Number of test packets to send.
nt_timeout Timeout value in seconds by which time the test packet sent must receive a response. Beyond this time, the linkloop attempt will abort.
nt_dest_macaddr MAC address of the destination interface.
nt_src_macaddr Source Mac address.
nt_macaddr_len Common length of the destination and source MAC addresses.

SEE ALSO

netmgr_linkloop(9F), netmgr_show_linkloop_stats(9F), subsystem_nwmgr(1M)
NAME

netmgr_nameval_t - All options in this structure are passed to the subsystem.

SYNOPSIS

#include <netmod.h>

DESCRIPTION

All options specified in command-line except attributes are passed to the subsystems in the form of this structure. The fields of this structure are populated from netmgr_option_t.

STRUCTURE MEMBERS

typedef struct {
    netmgr_nameval_t    *nv_next;
    char                *nv_name;
    uint64_t             nv_opt_int;
    char                *nv_cl_input;
    netmod_attrib_type_t nv_type;
    void                *nv_value
} netmgr_nameval_t;

nv_next Pointer to the next netmgr_nameval_t structure.

nv_name Multi-letter form of the option.

nv_opt_int Bit mapped integer equivalent of the option.

nv_cl_input String that contains the option as specified in command-line. This can be multi-letter, abbreviation, or single letter.

nv_type Type of the argument. If the option does not take an argument, it is set to NA_TYPE_NONE.

nv_value Contains the argument. If no argument is present, this field is set to NULL.

SEE ALSO

netmgr_cmd_t(9S), netmgr_find_opt(9F), netmgr_find_opt_for_int(9F), netmgr_find_opt_by_value(9F)
NAME
netmgr_option_t - Contains option related information

SYNOPSIS
#include <netmod.h>

DESCRIPTION
Structure to hold the complete information related to an option.

STRUCTURE MEMBERS
typedef struct {
    struct netmgr_option  *nt_next;
    char                  *nt_multi;
    char                   nt_abbr;
    char                   nt_single;
    netmgr_option_type_t   nt_option_type;
    uint64_t               nt_opt_int;
    netmgr_arg_req_type_t  nt_arg_req;
    netmod_attrib_type_t   nt_arg_type
}netmgr_option_t;

nt_next Pointer to the next option in the linked list.
nt_multi Multi-letter option string.
nt_abbr If option has an abbreviated form, this field contains the abbreviation string. If not, it holds an empty string.
nt_single Single letter equivalent of an option. If an option does not have a single letter equivalent, it contains NULL character.
nt_option_type This field indicates the enumerated option type to which this option belongs.
nt_opt_int The bit mapped integer equivalent for the option. Meant to ease option processing by avoiding the alternative of performing string compares.
nt_arg_req Indicates whether the option requires an argument or not.
nt_arg_type If the previous field, nt_arg_req, contains NM_REQ or NM_OPTIONAL, this field indicates the type of the argument. Otherwise, it contains NA_TYPE_NONE.

SEE ALSO
netmgr_nameval_t(9S), subsystem_nwmgr(1M)
**NAME**

netmgr_option_type_t - Indicates the option type

**SYNOPSIS**

```c
#include <netmod.h>
```

**Description**

Option types that will be indicated in `netmgr_option_t` structure by the field `nt_option_type`.

**STRUCTURE MEMBERS**

```c
typedef enum netmgr_option_type {
    NM_OPERATION = 1,
    NM_TARGET,
    NM_OP_QUAL,
    NM_TGT_QUAL,
    NM_ATTRIB,
    NM_UNKNOWN,
    NM_SUBSYS_QUAL
} netmgr_option_type_t;
```

- **NM_OPERATION** Specifies that the option type is an operation.
- **NM_TARGET** Specifies that the option type is a target.
- **NM_OP_QUAL** Specifies that the option type is an operational qualifier.
- **NM_TGT_QUAL** Specifies that the option type is a target qualifier.
- **NM_ATTRIB** Specifies that the option type is an attribute.
- **NM_UNKNOWN** Specifies that the option type is unknown.
- **NM_SUBSYS_QUAL** Specifies that the option type is a subsystem qualifier.

**SEE ALSO**

`subsystem_nwmgr(1M), netmgr_option_t(9S)`
netmgr_perform_cra(9F)

NAME

netmgr_perform_cra - Invokes the Critical Resource Analysis (CRA) framework to perform resource analysis on a set of hardware path

SYNOPSIS

#include <sys/cra.h>

int32_t netmgr_perform_cra(
    cra_event_t event,
    char *context,
    char *hw_paths[],
    char **buf
);

PARAMETERS

event Either ANALYSE, ANALYSE_LOCK, or RELEASE.
context One of the several context strings specified in the CRA framework Design Doc.
hw_paths List of hardware paths of interfaces for which CRA is to be performed. The list must be terminated by a NULL pointer.
buf Buffer containing the details of the CRA. The interpretation must be as specified in the CRA framework. The memory allocation for this buffer is done by the CRA framework. However, it is the responsibility of the calling subsystem to free the allocated memory.

DESCRIPTION

This function serves as a wrapper and invokes the CRA framework to perform critical resource analysis on a set of hardware paths. This routine must be invoked by subsystems that choose to perform implicit CRA before executing a destructive operation on its interfaces. This routine can also be invoked as part of preview for a destructive operation. For information about the CRA framework, see CRA framework Design Doc. This routine supports all the events and contexts identified by the CRA framework, because it is essentially a wrapper that invokes the cra_main API in the CRA framework shared library. The calling subsystem must ascertain the suitable event and the context based on its requirements.

RETURN VALUES

Positive integer Success. One of the enumerated values in cra_ret_t that indicates overall cumulative criticality of all the hardware paths passed as input.

Negative integer Error. Returns a negative value (-errno). This error code is defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

subsystem_nwmgr(1M)
netmgr_reset_if(9F)

NAME

netmgr_reset_if - Performs a hardware reset of the interface

SYNOPSIS

#include <netmod.h>

int32_t netmgr_reset_if(
    int32_t fd
);

PARAMETERS

fd File descriptor obtained through the netmgr_get_dlpi_fd service routine.

DESCRIPTION

This function invokes the HP-DLPI DL_HP_HW_RESET_REQ primitive to perform a hardware reset of the interface.

RETURN VALUES

0 Success
-1 Success

CONSTRAINTS

This API can be used by subsystems registered with HP-DLPI.

SEE ALSO

netmgr_reset_stats(9F)
NAME

netmgr_reset_stats - Resets the interface related statistics

SYNOPSIS

#include <netmod.h>

int32_t netmgr_reset_stats(
    int32_t fd
);

PARAMETERS

fd  File descriptor obtained through the netmgr_get_dlpi_fd service routine.

DESCRIPTION

This function resets the interface related statistics through the HP-DLPI
DL_HP_RESET_STATS_REQ primitive.

RETURN VALUES

0  Success

-1  Error

CONSTRAINTS

This API can be used by subsystems registered with HP-DLPI.

SEE ALSO

netmgr_reset_if(9F)
NAME

netmgr_set_ifadmin - Used to set the Administrative (ifAdmin) state of the interface

SYNOPSIS

#include <netmod.h>

int32_t netmgr_set_ifadmin(
    int32_t fd,
    int32_t mode
);

PARAMETERS

fd File descriptor obtained through the netmgr_get_dlpi_fd service routine.

mode Indicates the state to which the interface must be set. Set to 1 for UP and 2 for DOWN.

DESCRIPTION

This function invokes the HP-DLPI DL_HP_SET_IFADMIN_REQ primitive to set the Administrative (ifAdmin) state of the interface to UP or DOWN.

RETURN VALUES

0 Success

Negative integer Error. Negative value of the appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

This API can be used by subsystems registered with HP-DLPI.

SEE ALSO

netmgr_set_macaddr_to_fact(9F), netmgr_set_priv (9F), netmgr_set_macaddr(9F)
netmgr_set_macaddr(9F)

NAME

netmgr_set_macaddr - Sets or modifies the MAC address of an interface.

SYNOPSIS

#include <netmod.h>

int32_t netmgr_set_macaddr(
    int32_t fd,
    const u_char *macaddr,
    int32_t macaddr_len
);

PARAMETERS

fd File descriptor of a device file opened using the netmgr_get_dlpi_fd API.
macaddr The MAC address to which the interface MAC address must be set.
macaddr_len The length of MAC address in macaddr

DESCRIPTION

This function changes the MAC address of an interface to the address specified in macaddr.

RETURN VALUES

0 Success
-1 Error

CONSTRAINTS

This API can be used by subsystems registered with HP-DLPI.

SEE ALSO

netmgr_set_macaddr_to_fact(9F), netmgr_set_priv(9F), netmgr_set_ifadmin(9F)
netmgr_set_macaddr_to_fact(9F)

NAME

netmgr_set_macaddr_to_fact - Gets the factory MAC address of the interface and sets it as the current MAC address

SYNOPSIS

#include <netmod.h>

int32_t netmgr_set_macaddr_to_fact(
    int32_t fd
);

PARAMETERS

fd   File descriptor that was successfully returned in a previous netmgr_get_dlpi_fd call.

DESCRIPTION

This function obtains the factory MAC address of the interface and sets it as the current MAC address. The file descriptor must be opened and attached to an interface whose MAC address must be set.

RETURN VALUES

0      Success

Negative integer Error. Negative value. The appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

This API can be used by subsystems registered with HP-DLPI.

SEE ALSO

netmgr_set_macaddr (9F), netmgr_set.priv(9F), netmgr_set_ifadmin(9F)
netmgr_set_priv(9F)

NAME

netmgr_set_priv - Sets the privilege for the process

SYNOPSIS

#include <netmod.h>

int32_t netmgr_set_priv(
    const char *priv_set
);

PARAMETERS

priv_set Privilege set to which the process effective privilege must be upgraded.

DESCRIPTION

A network management specific authorization called hpux.network.config is created in the SAFeR database. This authorization is required to perform potentially destructive operations such as --set, --add, --delete, --enable, --disable, --reset. If the user specified operation is among the list of potentially destructive operations, the int32_t function must be invoked. The int32_t function checks whether the user is authorized to perform configuration related operations, by invoking the acps_simplecheckauth ACPS API. If the ACPS API returns ACPS_ALLOW, the effective privileges for the process is upgraded to the privilege set specified in the input parameter priv_set, by invoking the priv_set_effective API. A success value is returned only if both steps succeed. The caller must proceed with the user request based on the return value.

RETURN VALUES

0 Success

Negative value Negative value of the SAFeR defined error code.

CONSTRAINTS

None

SEE ALSO

netmgr_set_macaddr(9F), netmgr_set_macaddr_to_fact (9F), netmgr_set_ifadmin(9F)
NAME

netmgr_show_attrib_value - Displays attribute value of the structure.

SYNOPSIS

#include <netmod.h>

void netmgr_show_attrib_value(
    const netmod_attrib_t *a,
    const char *ifname,
    int32_t catalog_fd
);

PARAMETERS

a The attribute to be displayed.
if_name The interface name for which the attribute must be displayed.
catalog_fd File descriptor of the catalog file from which the attribute description string is displayed. If no catalog file is present or for scriptable display, the caller can set the parameter to -1.

DESCRIPTION

This function displays a single attribute given in the netmod_attrib_t structure. This function also displays errors for the previous failure on that attribute, if any.

RETURN VALUES

None

CONSTRAINTS

None

SEE ALSO

netmgr_show_union (9F), netmgr_show_attribs (9F)
NAME

netmgr_show_attribs - Displays attributes of an interface

SYNOPSIS

#include <netmod.h>

void netmgr_show_attribs(
    const char *ifname,
    const netmod_attrib_t *attrib_list,
    const char *attrib_set,
    uint64_t disp_flag
);

PARAMETERS

ifname The interface name for which the attribute listing must be displayed.

attrib_list The list of netmod_attrib_t structures which contain the attributes and
attribute specific values for the interface to be displayed.

attrib_set The name of the attribute set, such as current, saved, or any other named
attribute set. If this parameter is NULL, the default attribute set is current.

DESCRIPTION

This function displays the following:

- All the attributes of an interface given in the list of netmod_attrib_t structures.
- Errors, if any, for the previous failure on that attribute. -- attribute set such as current, saved, or any other named attribute set based on the contents of the input parameter, attrib_set

RETURN VALUES

None

CONSTRAINTS

None

SEE ALSO

netmgr_show_union (9F), netmgr_show_attrib_value(9F)
NAME
netmgr_show_cra - Invokes the CRA framework to perform critical resource analysis on a
hardware path

SYNOPSIS
#include <netmod.h>

int32_t netmgr_show_cra(
    char * hw_path,
    uint64_t disp_flag
);

PARAMETERS
hw_path Hardware path of interface for which CRA is to be performed.
disp_flag Flag to indicate the display format. Must be set to either NI_SCRIPT for scriptable
    format or 0 for readable format

DESCRIPTION
This function invokes the CRA framework to perform Critical Resource Analysis on a given
hardware path and displays the result. This function must be invoked by subsystems using the
-cra operation. The display format depends on the disp_flag input parameter. If the disp_flag is
set to NI_SCRIPT, the display will be in a scriptable format. Otherwise, it will be in a readable
format. This routine invokes the cra_main API of the CRA framework, with event set to ANALYZE
and context string set to CardDelete. The usage information returned by cra_main is displayed
without any further formatting.

RETURN VALUES
0 Success
Negative integer Error. Negative value of appropriate error code as defined in <sys/
    errno.h>.

CONSTRAINTS
None

SEE ALSO
netmgr_show_lan_if_list (9F), netmgr_show_lan_if_verbose (9F), netmgr_show_stats(9F)
netmgr_show_lan_if_list (9F)

NAME

netmgr_show_lan_if_list - Displays the fixed set of attributes

SYNOPSIS

#include <netmod.h>

int32_t netmgr_show_lan_if_list(
    const dl_hp_ext_ppa_info_t *ppa_list,
    const int32_t num_ppas,
    uint64_t disp_flag
);

PARAMETERS

ppa_list The list of dl_hp_ext_ppa_info_t structures that must be displayed.
num_ppas The number of PPAs that must be displayed.
disp_flag Flag to indicate the display format. Must be set to either NI_SCRIPT for scriptable format or 0 for readable format

DESCRIPTION

This function displays the fixed set of attributes for each of the PPAs represented by its dl_hp_ext_ppa_info_t structure in the ppa_list. The display format depends on the disp_flag input parameter. If disp_flag is set to NI_SCRIPT, the display will be in scriptable format. Otherwise, it will be in a tabular format.

RETURN VALUES

0 Success
Negative integer Error. Negative value of the appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

This API can be used by subsystems registered with HP-DLPI.

SEE ALSO

netmgr_show_lan_if_verbose(9F), netmgr_show_stats(9F), netmgr_show_cra (9F)
netmgr_show_lan_if_verbose(9F)

NAME

netmgr_show_lan_if_verbose - Displays the fixed set of attributes in verbose mode

SYNOPSIS

#include <netmod.h>

int32_t netmgr_show_lan_if_verbose(
  int32_t ppa,
  const netmod_attrib_t *attrib_list,
  const char *attrib_set,
  uint64_t disp_flag
) ;

PARAMETERS

ppa      The PPA for which the verbose listing is required.
attrib_list The netmod_attrib_t structure that contains the attributes and attribute specific values for the interface to be displayed.
attrib_set The name of the attribute set such as current, saved or any other named attribute set. If this parameter is NULL, the default attribute set is current.
disp_flag Flag to indicate the display format. Must be set to either NI_SCRIPT for scriptable format or 0 for readable format

DESCRIPTION

This function displays fixed set of attributes in the dl_hp_ext_ppa_info_t structure for the given PPA. The display format depends on the disp_flag input parameter. If the disp_flag is set to NI_SCRIPT, the display will be in a scriptable format. Otherwise, it will be in a readable format. This service does not display the autoneg attribute for virtual interfaces. If there are additional attributes to be displayed in addition to the fixed set, subsystems can provide them in the form of the attrib_list parameter.

RETURN VALUES

0         Success
Negative integer Error. Negative value of the appropriate error code as defined in <sys/errno.h>

CONSTRAINTS

This API can be used by subsystems registered with HP-DLPI.

SEE ALSO

netmgr_show_lan_if_list(9F), netmgr_show_stats(9F), netmgr_show_cra(9F)
NAME

netmgr_show_linkloop_stats(9F) - Performs linkloop diagnostics

SYNOPSIS

#include <netmod.h>

int32_t netmgr_show_linkloop_stats(
    netmgr_linkloop_t *linkloop_info,
    netmgr_linkloop_stats_t *linkloop_stats,
    uint64_t disp_flag
);

PARAMETERS

linkloop_info Structure containing all the necessary information for linkloop diagnostics to be performed. If the command line options contain combinations of --time, --interval, and --iterations, the caller can convert these into intervals and iterations. The conversion also includes assignment of default values for the three time related options

linkloop_stats Structure containing the number of test packets sent and received. The caller can deduce the percentage of packets dropped from these two, if required.

disp_flag Displays flag with NI_SCRIPT bit set for scriptable format and unset for readable format.

DESCRIPTION

This function performs linkloop diagnostics from a specific interface to the MAC address of the destination interface. The MAC address must be specified as an attribute.

RETURN VALUES

0 Success
Negative integer Error. Negative value of the appropriate error code as defined in <sys/errno.h>.

CONSTRAINTS

None

SEE ALSO

netmgr_linkloop(9F)
netmgr_show_stats(9F)

NAME
netmgr_show_stats - Gets statistics for a given interface

SYNOPSIS
#include <netmod.h>

int32_t netmgr_show_stats(
    int32_t fd,
    netmgr_stats_option_type_t stats_option,
    uint64_t disp_flag
);

PARAMETERS
fd File descriptor of the LAN instance that is opened and attached by using the netmgr_get_dlpi_fd service.
stats_option Argument to the --stats target qualifier. Enumerated value of type netmgr_stats_option_type_t.
disp_flag Flag to indicate the display format. Must be set to either NI_SCRIPT for scriptable format or 0 for readable format

DESCRIPTION
This function gets the statistics for the given instance based on the option specified in stats_option and displays it. If stats_option is set to NULL, the default is mibstats. The display format depends on the disp_flag input parameter. If the disp_flag is set to NI_SCRIPT, the display will be in a scriptable format. Otherwise, it will be in a readable format.

RETURN VALUES
Positive integer Success
Negative integer Error. Negative value of appropriate error code as defined in <sys/errno.h>

CONSTRAINTS
None

SEE ALSO
netmgr_show_lan_if_list (9F), netmgr_show_lan_ifVerbose(9F), netmgr_show_cra(9F)
NAME
netmgr_show_union - Displays the attribute value

SYNOPSIS
#include <netmod.h>

void netmgr_show_union(
    FILE *fp,
    netmod_data_type_t type,
    const char *display_fmt,
    netmod_union_t value
) ;

PARAMETERS
fp The file descriptor to which the attribute value must be written to.
    Usually set to either stdout or stderr.
type Data type of the attribute value to be displayed.
display_fmt The display format to be used to display the attribute value. Must be set to a
    format known to the fprintf system call.
value The netmod_union_t structure containing the attribute value that is
dereferenced based on the type parameter.

DESCRIPTION
This function displays a single attribute value given its data type and display format.

RETURN VALUES
None

CONSTRAINTS
None

SEE ALSO
netmgr_show_attrib_value (9F), netmgr_show_attribs (9F)
netmgr_stats_type_t(9S)

NAME

netmgr_stats_type_t - Specifies the option type of the statistics target qualifier

SYNOPSIS

#include <netmod.h>

DESCRIPTION

Option types that can be specified as argument to —stats target qualifier.

STRUCTURE MEMBERS

typedef enum {
    MIBSTATS = 1,
    MIBSTATS_64,
    MIBSTATS_DLPI,
    MIBSTATS_ALL,
    MIBSTATS_EXT,
    IF_SPECIFIC
} netmgr_stats_type_t;

MIBSTATS Used to get standard 32-bit MIB-II statistics.
MIBSTATS_64 Used to get standard 64-bit statistics.
MIBSTATS_DLPI Used to get HP-DLPI statistics.
MIBSTATS_ALL Used to get statistics for all the interfaces.
MIBSTATS_EXT Used to get statistics of all the options.
IF_SPECIFIC Used to get interface-specific 64-bit statistics.

SEE ALSO

subsystem_nwmgr(1M)
netmgr_update_version(9F)

NAME
netmgr_update_version - Detects the version number of nwmgr

SYNOPSIS
#include <sys/dlpi_ext.h>
int32_t netmgr_update_version(
    int32_t subsys_version
);               

PARAMETERS
subsys_version Version supported by the subsystem.

DESCRIPTION
This function can be used by the subsystems to detect the version number of nwmgr passed through the subsystem_name_netmgr_main routine. This entry routine is higher than the version number of the subsystem.

RETURN VALUES
0 Success
–1 Failure. The nwmgr core is unable to support the subsystem version.

CONSTRAINTS
None

SEE ALSO
subsystem_nwmgr(1M)
netmgr_validate_cmd(9F)

NAME

netmgr_validate_cmd - Validates the command line

SYNOPSIS

#include <netmod.h>

int32_t netmgr_validate_cmd(
    const netmgr_cmd_t *cmd,
    netmgr_policy_type_t policy_type,
    netmgr_policy_t policy[],
    uint32_t num_policies
);

PARAMETERS

cmd The command structure.

policy_type Type of policy. Valid values include: NM_POLICY_ERROR and
            NM_POLICY_WARN.

policy Array of bit masks, each representing a policy.

num_policies Number of entries in the array.

DESCRIPTION

This function validates the command line by applying the given set of policies to the command
structure, except for those that are marked with NI_IGNORE bit. If the policy type is ERROR,
generate error messages to stderr for policies that match with the command. If the policy type
is WARN, generate informative messages to stderr for matching policies that some options are
being ignored. A policy is a bit mask, with one bit for each UCLI option. The policy bit mask
indicates which option or option combination is considered erroneous or superfluous. The
--time, --interval and --iteration options are treated as a single group, and are specified
with the bit NI_TIME. There is a special bit, named NI_IGNORE, which indicates that the
associated policy must be ignored. The overall approach is to indicate what options are
unsupported, rather than what are supported. This helps in generating precise error or warning
messages. This bit mask model is expected to handle many policies, but not all. Some examples
of policies that do not fit the model are as follows:

• Policies that require knowing how many times an option occurred on the command line.
  For example, check that there is exactly one --class_instance option in the command.

• Policies that treat many options as equivalent. For example, check that at least one among
  --instance and --class_instance is specified. Policies that involve option arguments.
  For example check that an option has at least one argument.

RETURN VALUES

0 No error policies matched with the command line. If policy_type is NM_POLICY_WARN,
the return code is always 0.

ENOTSUP One or more error policies matched with the command.

CONSTRAINTS

None

SEE ALSO

subsystem_nwmgr(IM)
NAME

netmgr_verify_one_instance - Verifies if the command line specifies one network interface

SYNOPSIS

#include <netmod.h>

uint32_t netmgr_verify_one_instance(
    const netmgr_cmd_t *cmd
);

PARAMETERS

cmd  The command structure.

DESCRIPTION

This function verifies if the command line specifies exactly one network interface. If not, it
generates an error message to stderr. The interface can be specified with a class-instance, a
class and an instance, or some other equivalent.

RETURN VALUES

1  True, that is, only one interface is specified in the command.
0  False, that is, either no interface or more than one interface is specified by the command.

CONSTRAINTS

None

SEE ALSO

subsystem_nwmgr(1M)
NAME

netmod_attrib_t - Provides attribute related information

SYNOPSIS

#include <netmod.h>

DESCRIPTION

Structure to hold information related to an attribute.

STRUCTURE MEMBERS

typedef struct {
    netmod_attrib_t* na_next;
    netmod_string_t na_name;
    char* na_conf_name;
    netmod_data_type_t na_type;
    netmod_granularity_t na_gran;
    netmod_union_t na_default;
    netmod_union_t na_min;
    netmod_union_t na_max;
    netmod_union_t na_value;
    netmod_union_t na_old_value;
    netmod_union_t* na_value_set;
    uint32_t na_value_set_size;
    netmod_attrib_flags_t na_flags;
    char* na_display_fmt;
    int32_t na_errno;
    char* na_msg;
    int32_t na_msg_setid;
    int32_t na_msgmsgid;
    void* na_lib_priv1;
    void* na_lib_priv2;
    netmod_union_t na_lib_priv3;
    void* na_rsvd1;
    int64_t na_rsvd2
} netmod_attrib_t;

na_next

Pointer to the next group in the linked list.

na_name

Names for this attribute (internal and external). The internal
name and the name in the CLI for the attribute must be identical.

na_conf_name

Name of the attribute in the configuration file. It can be NULL
if the name in the configuration file can be derived from
na_name.nnt_name using standard rules.

na_type

Data type of the attribute.

na_gran

Granularity (discrete, continuous, and so on).

na_default

Default value (the value applied by the driver when the interface
is first initialized). Read-only. Actual data type depends on
na_type.

na_min

Minimum value for range-bound attributes.

na_max

Maximum value for range-bound attributes.

na_value

Attribute value in the requested attribute set (current or saved).

na_old_value

Value of the attribute before a set operation.

na_value_set

Array of possible values for discrete-value attributes.

na_value_set_size

Number of possible values for discrete-value attributes.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>na_flags</td>
<td>Bit mask.</td>
</tr>
<tr>
<td>na_display_fmt</td>
<td>Format to be used to display attribute value that can be used directly in an <code>fprintf</code> call.</td>
</tr>
<tr>
<td>na_errno</td>
<td>Error code from <code>errno.h</code> if the operation on the attribute was not successful.</td>
</tr>
<tr>
<td>na_msg</td>
<td>String containing the message that the subsystem shared library would like <code>nwmgr</code> to display. It can be an error message or an informative one.</td>
</tr>
<tr>
<td>int32_t na_msg_setid</td>
<td>Set ID of the message in the catalog file for localization purposes.</td>
</tr>
<tr>
<td>na_msgmsgid</td>
<td>Message ID of the message in the catalog file for localization purposes.</td>
</tr>
<tr>
<td>na_lib_priv1</td>
<td>Reserved field for subsystem specific use. Must be set to NULL. Do not use this field.</td>
</tr>
<tr>
<td>na_lib_priv2</td>
<td>Reserved field for subsystem specific use. Must be set to NULL. Do not use this field.</td>
</tr>
<tr>
<td>na_lib_priv3</td>
<td>Reserved field for subsystem specific use. Must be set to 0. Do not use this field.</td>
</tr>
<tr>
<td>na_rsvd1</td>
<td>Reserved field. Modules must set this to NULL.</td>
</tr>
<tr>
<td>na_rsvd2</td>
<td>Reserved field. Modules must set this to 0.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`netmgr_cmd_t(9S), netmgr_show_attrib_value(9F), netmgr_show_attribs(9F),
netmgr_show_lan_if_verbose(9F), netmgr_conf_get_indices_for_nameval(9F),
netmgr_conf_get_indices_for_name(9F), netmgr_clone_attrib(9F), netmgr_free_cloned_attrib(9F),
netmgr_conf_get_val_for_name_idx(9F), netmgr_conf_get_atlist_for_idx(9F),
netmgr_conf_set_atlist_for_idx(9F), netmgr_conf_delete_nameval_for_idx(9F),
netmgr_conf_delete_attrib(9F), netmgr_conf_set_attrib_value(9F), subsystem_netmgr(1M)"
NAME
netmod_freefunc_t - Free function pointer for each API.

SYNOPSIS
#include <netmod.h>

DESCRIPTION
This is a type definition for a free-function pointer. Subsystems can define a free-function for each API and pass them to the NIC tool. The NIC tool must invoke this function after using the data obtained through that API.

STRUCTURE MEMBERS
typedef void (*netmgr_freefunc_t) (uint32_t version, void * ptr);

CONSTRAINTS
The NIC tool must invoke the free-function only if the pointer is not NULL. The NIC tool must pass the version number that the subsystem returned through the API corresponding to this free-function.

SEE ALSO
subsys_get_subsys_tasks(9F), subsys_get_first_page_data(9F), subsys_get_if_tasks(9F),
subsys_get_if_details(9F), subsys_get_task_info(9F), subsys_do_task(9F), subsys_get_cmd_preview(9F),
subsys_refresh_task_info(9F)
NAME
netmod_granularity_t - Specifies the granularity of attribute values

SYNOPSIS
#include <netmod.h>

DESCRIPTION
Enumerated data type to represent the granularity of the attribute values.

STRUCTURE MEMBERS
typedef enum netmod_granularity {
    NA_GRAN_NONE,
    NA_GRAN_DISCRETE,
    NA_GRAN_RANGE_BOUNDED
} netmod_granularity_t;

NA_GRAN_NONE Sets the granularity value of the attribute to none.
NA_GRAN_DISCRETE Sets the granularity value of the attribute to discrete. For example, the attribute values can be 0, 4, 16, 32.
NA_GRAN_RANGE_BOUNDED Sets the granularity value of the attribute to range bound. For example, the attribute values can be (257 to 1500), (1023 to 9000).

SEE ALSO
netmod_attrib_t(9S), subsystem_nwmgr(1M)
netmod_group_flags_t (9S)

NAME

netmod_group_flags_t - Enumeration data type (bit flag) for group information.

SYNOPSIS

#include <netmod.h>

DESCRIPTION

This is a bit mask whose fields can be OR’d together. The bits are set by the subsystem. The API caller can only read these flags.

STRUCTURE MEMBERS

typedef enum netmod_group_flags {
    NG_FLAG_KEEP_TOGETHER = 0x01,
    NG_FLAG_USE_FEW_ROWS = 0x02,
    NG_FLAG_HEADER = 0x04
} netmod_group_flags_t;

NG_FLAG_KEEP_TOGETHER The next group must be placed in the same expand/collapse button (in the HTML page) or TUI page as this group. If there is no next group, this is a no-op.

NG_FLAG_USE_FEW_ROWS Keep as many attributes of this group in one row as possible, subject to other considerations such as minimizing scrolling and general GUI design policies. Valid only for groups of type NG_TYPE_ATTRIB.

NG_FLAG_HEADER The leading group in a task page, which contains information that can help orient the user. It is good practice for subsystems to use such a header group in all the task pages. The NIC tool must display this group distinctively. The header group, if present, must be the first group in its task and the NIC tool must treat all attributes as Read-Only.

SEE ALSO

netmod_group_t(9S)
NAME
netmod_group_t - Details of a group.

SYNOPSIS
#include <netmod.h>

DESCRIPTION
A group represents a collection of related data, such as attributes, range, file name, or the list of
interfaces needed for creating and modifying virtual interfaces (see netmod_group_type_t).

NOTE: It is possible for the subsystem library to define a group that contains only ng_msgs
and no ng_data.

STRUCTURE MEMBERS

typedef struct netmod_group {
    netmod_group_t * ng_next;
    char * ng_display_name;
    void * ng_data;
    uint64_t ng_data_len;
    netmod_group_type_t ng_type;
    netmod_group_flags_t ng_flags;
    netmod_string_t * ng_msgs;
    uint32_t ng_setid;
    uint32_t ng_display_name_msgid;
    void * ng_rsvd;
} netmod_group_t;

ng_next Pointer to the next group in the linked list.

ng_display_name Displayed name of the group. (There is no internal name for
groups.) May be NULL. However, if the group could be
condensed into an Expand/Collapse button in the NIC tool,
the name must not be NULL.

ng_data Pointer to attribute list or interface list, depending on the group
type.

ng_data_len Length of the data, that is, the number of attributes or number
of interface lists, depending on group type.

ng_type Type of contents in this group. Can be attributes, range, file
name, or interface list (for creating/modifying virtual interfaces).

ng_flags Bit mask.

ng_msgs NULL-terminated list of localizable messages. The
ns_display_name field contains the message and the
associated fields provide localization data. Other fields are
irrelevant.

ng_setid Set ID in the catalog for this group.

ng_display_name_msgid Message ID in the catalog for the display name.

ng_rsvd Reserved field. Subsystems must set this to NULL in the first
release. The API caller must not use it or change it in the first
release.

SEE ALSO
netmod_group_type_t(9S), netmod_group_flags_t(9S), netmod_string_t(9S)
NAME

netmod_group_type_t - Enumerate data type for group.

SYNOPSIS

#include <netmod.h>

DESCRIPTION

This enumeration represents the group type, that is, the type of contents of a group.

STRUCTURE MEMBERS

typedef enum netmod_group_type {
    NG_TYPE_ATTRIB,
    NG_TYPE_IF_LIST,
    NG_TYPE_FILENAME,
    NG_TYPE_RANGE
} netmod_group_type_t;

NG_TYPE_ATTRIB Attributes.
NG_TYPE_IF_LIST List of interfaces for creating/modifying virtual interfaces.
NG_TYPE_FILENAME Name of a file containing the text that must be displayed as is. For example, CRA output.
NG_TYPE_RANGE Range table.

SEE ALSO

netmod_group_t(9S)
NAME

netmod_range_table_t – Indicates the range of the table listing the interfaces.

SYNOPSIS

#include <netmod.h>

DESCRIPTION

This structure addresses the scenario in which there is a quantity R, whose values are within a range (minimum and maximum). The values might be integers or floats. The set of values of R is divided into N ranges, each with a start and end, such that the N ranges together include every value of R without duplicates. That is, the ranges do not overlap and also do not have any gaps between them.

There is another quantity Q that is dependent on the value of R. For each range of R, quantity Q needs to be set to a value. There might be more than one such dependent quantity. There might be zero or more attributes that are related to R and its dependents in some way.

The number of ranges, N, in this scenario and the number of dependent quantities, D, in this scenario are determined by the subsystem library.

The user specifies the following:

• End values for each range of R
• The value of each dependent quantity for each range

The driver shared library must do the following irrespective of whether the library internally handles R or Q as a single list-valued attribute:

• Each range of R, except the last, must be represented as a separate netmod_attrib_t with the same data type, granularity, minimum, and maximum as R. (The last range is implicit because its starting value is the same as the ending value of the previous range, and its end is the same as the maximum of R.) The library must provide a default value for each of these \(N-1\) attributes. The value of each attribute is the ending value for its range.

• Each dependent quantity Q must also be represented as N separate netmod_attrib_t structures, each with the same data type, granularity, minimum, and maximum as Q. These must also have appropriate defaults.

The NIC tool must do the following, with possible exceptions in the TUI:

• The values of R must be presented as a table with one row for each range, and two columns, one for the start values of the range and the other for the end values.

• The column for the end values must be edit enabled, except for the last row, which will contain the maximum value of R as a read-only value.

• The column for the start values must be read-only. In the first row, the start values must be the same as the minimum of R. In every other row, the NIC tool must ensure that the start value is the same as the end value of the previous row. As the user updates an end value, the NIC tool must update the start value for the next row.

• The range column must carry a header caption that is provided by the subsystem library.

• Each dependent quantity Q must be displayed as a column of editable attributes, with a header caption provided by the library.

• The columns for the range must be visually distinct from the columns for the dependent quantities.

• Handling more than one dependent quantity is a highly desired, though not mandatory, requirement in the first release.

• The set of related attributes, if any, must be displayed above the range table.
typedef struct netmod_range_table {
    netmod_string_t *nrt_col_headers;
    netmod_attrib_t *nrt_ranges;
    netmod_attrib_t **nrt_dependents;
    netmod_attrib_t *nrt_related;
    uint32_t nrt_num_ranges;
    uint32_t nrt_num_dependents;
    uint32_t nrt_num_related;
    void * nrt_rsvd;
} netmod_range_table_t;

**nrt_col_headers**
Captions for the range column and the dependent columns, including the help text. The range name must be the first one in the list, and the dependent attribute names shall be in the same order as in *nrt_dependents*.

**NOTE:** The length of this list is the same as 1 + *nrt_num_dependents*.

**nrt_ranges**
List of attributes, each representing one range except the last range. The length of the list is *nrt_num_ranges* – 1.

**nrt_dependents**
Array of linked lists of attributes. Each list corresponds to one dependent quantity Q. The length of each list is *nrt_num_ranges*.

**nrt_related**
List of attributes, possibly empty, that are related to the table. The list may not be NULL-terminated. The API caller must rely on the count in the *nrt_num_related* field.

**nrt_num_ranges**
Number of ranges for the range quantity R.

**nrt_num_dependents**
Number of dependent quantities Q.

**nrt_num_related**
Number of related attributes.

**nrt_rsvd**
Reserved. Set to NULL.

**SEE ALSO**

*netmod_attrib_t(9S), netmod_string_t(9S)*
NAME

netmod_row_t - Details of a row in the interface list.

SYNOPSIS

#include <netmod.h>

DESCRIPTION

Per-interface data for the list of interfaces provided in the table structure. The data includes fields required for virtual groups, such as membership flags. These fields are not relevant for the get_first_page_data routine.

STRUCTURE MEMBERS

typedef struct netmod_row {
    netmod_row_t    *nr_next;
    netmod_attrib_t *nr_attribs;
    uint32_t (boolean) nr_lib_membership_flag;
    uint32_t (boolean) nr_membership_flag;
    uint32_t          nr_compatibility_id;
    char              nr_if_name;
    netmod_row_flags_t nr_flags;
    uint32_t          nr_indent_level;
    void              *nr_rsvd;
} netmod_row_t;

nr_next
    Pointer to the next netmod_row_t structure.

nr_attribs
    Pointer to the list of attributes for this row. The first entry in the list must be a pseudo-attribute in which the na_name.ns_name field is if_name, the na_value field is a string containing the interface name, and the na_type field is ND_TYPE_STRING. Other fields in the pseudo-attribute are not relevant to the API. The number of attributes in each row, including the pseudo-attribute, is conveyed through the nt_num_columns field in the netmod_table_t.

nr_lib_membership_flag
    Library-private field. Can be used by the library before the call to get_task_info() to record whether a given interface earlier belonged to a virtual group.

nr_membership_flag
    Specifies whether the given interface belongs to the virtual group selected by the user. The NIC tool must update this field based on user input.

nr_compatibility_id
    Specifies the compatibility of this link with other interfaces in the list. Interfaces with same compatibility id value are compatible with each other. The NIC tool will use this information to display only compatible interfaces on the screen. If the value of this id is —1, then it is by default compatible with other ids.

nr_if_name
    Specifies the interface name.

nr_flags
    Bit mask.

nr_indent_level
    Specifies the indent level of the column. When the value is 0, the first column is not indented. When the value is 1 or higher, the first column after the check box is proportionately indented to reflect a hierarchical relationship with the previous element. This is needed for tables that display an interface in
each row. The amount of indentation is left to the NIC tool front end.
Reserved. Set to NULL in the first release.

nr_rsvd

SEE ALSO

netmod_table_t(9S), netmod_attrib_t(9S)
NAME
netmod_string_t - Holds the name of an attribute and supports localization

SYNOPSIS
#include <netmod.h>

DESCRIPTION
Structure to hold the name of an attribute. The structure supports localization.

STRUCTURE MEMBERS

typedef struct {
    netmod_string_t *ns_next;
    char *ns_name;
    char *ns_display_name;
    char *ns_desc;
    int32_t ns_code;
    uint32_t ns_set_id;
    uint32_t ns_display_namemsgid;
    uint32_t ns_descmsgid;
    char ns_mnemonic;
    void *ns_rsvd
}netmod_string_t;

ns_next
Pointer to the next entry in the list of netmod_name_t, or NULL if there are no more entries.

ns_name
Attribute name used for command-line display.

ns_display_name
Name to be displayed.

ns_desc
Description or help text for the attribute name.

ns_code
Integer code for the attribute. Used by SMH.

ns_set_id
Set in the message catalog where the display name and description occur.

ns_display_namemsgid
Message ID in the catalog for display name.

ns_descmsgid
Message ID in the catalog for description.

ns_mnemonic
Mnemonic letter for a task. Used by SMH.

ns_rsvd
Reserved. Modules must set this to NULL during first release.

SEE ALSO
netmod_attrib_t(9S), subsystem_nwmgr(1M)
**NAME**

netmod_task_info_flags_t - Enumeration data type for task information.

**SYNOPSIS**

```c
#include <netmod.h>
```

**DESCRIPTION**

This is a bit mask whose fields can be or'd together. The bits are set by the subsystem. The API caller can only read these flags.

**STRUCTURE MEMBERS**

```c
typedef enum netmod_task_info_flags {
    NTI_FLAG_DELEGATE_PREVIEW = 0x01,
    NTI_FLAG_NO_CMD_PREVIEW = 0x02,
    NTI_FLAG_SUBSYS_TASK = 0x04
} netmod_task_info_flags_t;
```

- **NTI_FLAG_DELEGATE_PREVIEW**: The NIC tool must generate the command Preview. The NIC tool uses a standard algorithm to generate the command Preview. Subsystems must appropriately populate all the data structure fields that are used in the algorithm. For more information on the command preview algorithm, see the “Supporting the HP SMH NIC Tool in LAN Drivers” chapter in the *HP-UX 11i v3 Driver Development Guide*.

- **NTI_FLAG_NO_CMD_PREVIEW**: Command Preview is not supported for this task. The NIC tool must not display the command Preview button when this flag is set.

  **NOTE:** Use this bit only as a last resort. Command Preview is a useful feature for users. It must be mandatory for all subsystems to provide this feature.

- **NTI_FLAG_SUBSYS_TASK**: Indicates a subsystem-wide task as opposed to a task that acts on a specific interface. For example, the task to create a new APA aggregate.

**SEE ALSO**

- *netmod_task_info*(9S)
NAME

netmod_task_info_t - Details of a task.

SYNOPSIS

# include <netmod.h>

DESCRIPTION

The task info structure encapsulates a task that the API caller can initiate.

NOTE: The NIC tool will display the qualifiers at the bottom of the task page, preceded by the title Configuration Options and followed by standard buttons such as OK. Subsystems must place different qualifier types such as task qualifiers and target qualifiers in separate groups with appropriate group names.

Subsystems may choose to have a header for a task page to orient the user. The header is an attribute group, usually with standard read-only attributes such as Hardware Path and Interface Description string. Subsystems must place the header group, if any, as the first group in nti_groups list.

STRUCTURE MEMBERS

typedef struct netmod_task_info {
    netmod_string_t nti_name;
    netmod_group_t *nti_groups;
    uint32_t nti_num_groups;
    netmod_group_t *nti_qual_groups;
    uint32_t nti_num_qual_groups;
    netmod_group_t *nti_msgs;
    netmod_task_info_flags_t nti_flags;
    char *nti_help_file;
    char *nti_help_index;
    uint32_t nti_errno;
    void *nti_rsvd;
} netmod_task_info_t;

nti_name

Name structure of the task. This is the same as the name provided to the API caller through subsys_get_*_tasks.

ns_name

This can contain the nwmgr options needed to generate the command preview.

ns_display_name

This is the name seen by the user.

ns_code

The task can be one among the standard codes, or the drivers can have subsystem specific task codes. For common tasks, the task code in the ns_code field can be one of the following values:

• NC_TASK_MOD_CURRENT_ATTRIBS
• NC_TASK_MOD_SAVED_ATTRIBS
• NC_TASK_RESET_IF
• NC_TASK_CREATE_MOD_VI
• NC_TASK_DELETE_VI
• NC_TASK_CHECK_CONN
• NC_TASK_FREEZE_SUBSYS
· NC_TASK_ERASE_SAVED_IF
· NC_TASK_TRANSFER_ATTRIBS
· NC_TASK_GET_CRA_USAGE
· NC_TASK_MOD_SUBSYS_CURRENT_ATTRIBS
· NC_TASK_MOD_SUBSYS_SAVED_ATTRIBS
· NC_TASK_DOWNLOAD_FW

For more information on the task codes, see task code.

**ns_desc**
Help Text for the task.

**ns_mnemonic**
One-character mnemonic for the task, to be used in the TUI.

**nti_groups**
Pointer to linked list of groups. May not be NULL-terminated.

**nti_num_groups**
Number of groups in **nti_group** list. The API caller must depend only on this number, and not on NULL termination.

**nti_qual_groups**
List of groups containing qualifiers, such as task qualifiers, target qualifiers, or subsystem qualifiers. Usually, these groups contain attributes or localized strings. The list may not be NULL-terminated.

**nti_num_qual_groups**
The number of qualifier groups.

**nti_msgs**
NULL-terminated list of messages to be displayed. The message may communicate an error, warning, or information. The ns_display_name contains the message and the associated fields provide localization data; other fields are irrelevant.

**nti_flags**
Bit mask

**nti_help_file**
Name of the GUI/TUI help files for this task. The help file name in the **nti_help_file** field must not include the suffix. If the subsystem delivered the help file to a directory within the standard SMH help directories, the relative path must be included. For example, apa_help or lacp/apa_help

**nti_help_index**
Name of the HTML index in the help file, without the leading #.

**nti_errno**
Error code from **errno.h**, if any error occurred while executing the task.

**nti_rsvd**
Reserved field. Subsystems must set this to NULL in the first release. The API caller must not use it or change it in the first release.
### Table 4-2 Common Tasks and Task Codes

<table>
<thead>
<tr>
<th>Task Code Name</th>
<th>Task Code</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC_TASK_MOD_CURRENT_ATTRIBS</td>
<td>1</td>
<td>‘a’</td>
<td>View/Modify current attributes of the selected interface. The current attributes are those that pertain to the running system, as opposed to next-boot values or defaults. The interface may be a virtual one, such as an APA aggregate or RDMA Virtual Group. <strong>NOTE:</strong> Modification involves updating the entire interface’s configuration: Current, Saved, or both, as specified by the user.</td>
</tr>
<tr>
<td>NC_TASK_MOD_SAVED_ATTRIBS</td>
<td>2</td>
<td>‘p’</td>
<td>View/Modify saved (persistent) attributes of the selected interface. Symmetric to Task Code 1.</td>
</tr>
<tr>
<td>NC_TASK_RESET_IF</td>
<td>4</td>
<td>‘r’</td>
<td>Reset the interface or some component of it. The task_info structure may include subsystem qualifiers for resetting specific components of the interface instead of the entire interface.</td>
</tr>
<tr>
<td>NC_TASK_CREATE_MOD_VI</td>
<td>6</td>
<td>‘c’</td>
<td>Create or modify a virtual interface.</td>
</tr>
<tr>
<td>NC_TASK_DELETE_VI</td>
<td>7</td>
<td>‘d’</td>
<td>Delete a virtual interface.</td>
</tr>
<tr>
<td>NC_TASK_CHECK_CONN</td>
<td>8</td>
<td>‘k’</td>
<td>Check connectivity through an interface, at one or more layers of the network stack.</td>
</tr>
<tr>
<td>NC_TASK_FREEZE_SUBSYS</td>
<td>9</td>
<td>‘f’</td>
<td>Freeze, that is, save the current configuration of a subsystem. <strong>NOTE:</strong> Freezing an interface is part of the NC_TASK_CURRENT_ATTRIBS with the ‘Saved’ qualifier.</td>
</tr>
<tr>
<td>NC_TASK_ERASE_SAVED_IF</td>
<td>10</td>
<td>‘e’</td>
<td>Erase (delete) the saved configuration for an interface.</td>
</tr>
<tr>
<td>NC_TASK_TRANSFER_ATTRIBS</td>
<td>11</td>
<td>‘t’</td>
<td>Transfer default values of attributes to the current set, for an interface. In the future, this could mean transfer an interface configuration from one attribute set to another, such as from default to current or current to saved. This is a super set of task code 1.</td>
</tr>
<tr>
<td>NC_TASK_GET_CRA_USAGE</td>
<td>12</td>
<td>‘u’</td>
<td>Show the CRA information i.e., the usage for an interface.</td>
</tr>
<tr>
<td>NC_TASK_MOD_SUBSYS_CURRENT_ATTRIBS</td>
<td>13</td>
<td>‘g’</td>
<td>View/Modify current global attributes of the subsystem. Symmetric to task code 1.</td>
</tr>
</tbody>
</table>
### Table 4-2 Common Tasks and Task Codes (continued)

<table>
<thead>
<tr>
<th>Task Code Description</th>
<th>Task Code</th>
<th>Task Code Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View/Modify saved global attributes of the subsystem. Symmetric to task code 2.</td>
<td>NC_TASK_MOD_SUBSYS_SAVED_ATTRIBS</td>
<td>14 'G'</td>
<td>View/Modify saved global attributes of the subsystem. Symmetric to task code 2.</td>
</tr>
<tr>
<td>Download NIC/HCA firmware.</td>
<td>NC_TASK_DOWNLOAD_FW</td>
<td>15 'w'</td>
<td>Download NIC/HCA firmware.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`netmod_group_t(9S), netmod_task_info_flags_t(9S)`
NAME

netmod_tab_code_t - Enumerates the code of the tab heading of the interface.

SYNOPSIS

#include <netmod.h>

DESCRIPTION

This enumeration represents the integer codes for the various tab areas in the NIC tool, such as NIC, RDMA, APA, and VLAN. When the NIC tool invokes an API that has a tab code parameter, the NIC tool must pass in the code for the tab from which the API is called. When applications other than the NIC tool use any such API, they must specify the code as NONE. The NIC tool libraries must be able to handle either case.

STRUCTURE MEMBERS

typedef enum netmod_tab_code {
    NTC_NONE,
    NTC_NIC,
    NTC_APA,
    NTC_VLAN,
    NTC_RDMA
} netmod_tab_code_t;

NTC_NONE No tab. Used only by applications other than the NIC tool.

NTC_NIC NIC tab

NTC_APA APA tab.

NTC_VLAN VLAN tab.

NTC_RDMA RDMA tab. Includes InfiniBand and the RDMA component of RNICs.

SEE ALSO
netmod_table_t(9S)

NAME

netmod_table_t - List of interfaces in table format.

SYNOPSIS

#include <netmod.h>

DESCRIPTION

List of interfaces in tabular format. If ng_group_type in netmod_group_t is set to NG_TYPE_IF_LIST, then ng_data will point to the data structure of this type.

STRUCTURE MEMBERS

typedef struct netmod_table {
    netmod_table_t      *nt_next;
    netmod_row_t        *nt_rows;
    netmod_string_t     *nt_col_headers;
    uint32_t             nt_num_rows;
    uint32_t             nt_num_columns;
    uint32_t             nt_max_members;
    netmod_table_flags_t nt_flags
    void                *nt_rsvd;
} netmod_table_t;

nt_next
    Next table in a linked list.

nt_rows
    Linked list of row structures, each containing the list of attributes for a specific row, starting with the interface name.

nt_col_headers
    List of column headers, each with a display name, description, and related localization data. Other fields in the name structure are not relevant.

nt_num_rows
    Number of rows (interfaces) in the table.

nt_num_columns
    Number of columns (attributes displayed per interface in the list) in the table, including the pseudo-attribute for interface name.

nt_max_members
    Maximum number of members that can be selected by the user. The field is not relevant in the context of get_first_page_data().

nt_flags
    Bit mask.

nt_rsvd
    Reserved. Set to NULL in the first release.

SEE ALSO

netmod_row_t(9S), netmod_attrib_t(9S)
NAME

netmod_union_t - Specifies the data types of attribute values

SYNOPSIS

#include <netmod.h>

DESCRIPTION

This union contains all possible data types of the value of an attribute. It also includes most of the data types specified in netmod_data_type_t and fields such as void *.

STRUCTURE MEMBERS

typedef union {
    void     *nu_void,
    int64_t   nu_int64,
    uint64_t  nu_uint64,
    double   nu_double,
    int 64_t  nu_bool,
    char     *nu_str,
    int32_t   nu_int32[2],
    uint32_t  nu_uint32[2]
} netmod_union_t

nu_void          Used when the attribute data type is ND_TYPE_ATTRIB, or if the data type is not included in netmod_data_type_t.
nu_int64         Used if the data type is 64-bit signed integer.
nu_uint64        Used if the data type is 64-bit unsigned integer.
nu_double        Used if the data type is floating point.
nu_bool          Used for ND_TYPE_BOOL.
nu_str           Used for ND_TYPE_STRING.
nu_int32[2]      Used if the data type is 32-bit signed integer. The preprocessor token INDEX32 specifies the element of the array to use.
nu_uint32[2]     Used if the data type is 32-bit unsigned integer. The preprocessor token INDEX32 specifies the element of the array to use.

SEE ALSO

netmod_attrib_t(9S), netmgr_show_union (9F), subsystem_nwmgr(1M)
new_pri_info_t(9S)

NAME
new_pri_info_t - Data type for the DL_HP_OOP_TOS OOP type.

SYNOPSIS
#include <sio/dlpi_ext.h>

DESCRIPTION
Contains the ToS. This OOP type is defined in dl_hp_oop_type_t.

STRUCTURE MEMBERS
typedef struct new_pri_info {
    uint8_t reserved[3];
    uint8_t ip_precedence;
} new_pri_info_t;
reserved Reserved for future use.
ip_precedence IP precedence communicated between DLPI and the IP layer.

SEE ALSO
dl_hp_oop_type_t(9S)
pri_override_t(9S)

NAME

pri_override_t - Types of priority override for a VLAN.

SYNOPSIS

#include <sio/dlpi_ext.h>

DESCRIPTION

The types of priority override for a VLAN are listed in this enumeration. Used by create and modify VLAN user-level requests and by drivers using the DL_PROP_VLAN_INFOP property.

STRUCTURE MEMBERS

typedef enum pri_override {
    OUT_CONF_PRI = 1,
    OUT_IP_HEADER,
    OUT_CONF_TOS
} pri_override_t;

OUT_CONF_PRI User-specified priority will be used (System default setting, also the only setting for non-IP).

OUT_IP_HEADER IP header ToS will be converted to 802.1p priority.

HP_DLPI_VLAN_TOS_SET VLAN ToS is set.

OUT_CONF_TOS User-specified ToS will be converted to 802.1p priority.

HP_DLPI_VLAN_PRI_OVERRIDE_SET VLAN priority override is set.

SEE ALSO

dl_hp_create_vlan_req_t(9S), dl_hp_modify_vlan_req_t(9S), tos_override_t(9S)
NAME

set_up_8022 - Sets up global information for the filter and formatting functions.

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int set_up_8022(
    u_char *buf_ptr,
    int len,
    u_char *dst_addr,
    u_char *src_addr
);

DESCRIPTION

The set_up_8022 function sets up global information used by the filter_packet,
format_link_nice, format_link_raw, and format_link_terse filter and formatting
functions. This routine walks through the buffer, pointing to 802.2 data, and copies protocol
header information to the appropriate global variables used by the filter and formatters
 Call this routine for each PDU_IN or PDU_OUT trace event.

PARAMETERS

buf_ptr Pointer to the beginning of the 802.2 information. It must not include MAC
 information.

len Length of the buffer, excluding the MAC header.

dst_addr Pointer to the 6-byte destination MAC address, extracted by local methods from
 the MAC header.

src_addr Pointer to the 6-byte source MAC address, extracted by local methods from the
 MAC header.

RETURN VALUES

Always returns 0.

CONSTRAINTS

None

SEE ALSO

set_up_ether(9F), set_up_ip(9F), set_up_link(9F)
set_up_ether(9F)

NAME

set_up_ether - Sets up global information for the filter and formatting functions.

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int set_up_ether(
    u_char *buf_ptr,
    int len,
    u_char *dst_addr,
    u_char *src_addr,
    int ether_type
);

DESCRIPTION

The set_up_ether function sets up global information used by the filter_packet,
format_link_nice, format_link_raw, and format_link_terse filter and formatting
functions. This routine walks through the buffer and copies protocol header information to the
appropriate global variables used by the filter and formatter routines. Call this routine for each
PDU_IN and PDU_OUT trace event that contains Ethernet packets.

PARAMETERS

buf_ptr Pointer to the beginning of the Ethernet data. It must not include the destination
address, source address, or Ethernet type information.

len Length of the buffer, excluding destination, source, and Ethernet type.

dst_addr Pointer to the 6-byte destination MAC address, extracted by local methods from
the MAC header.

csrc_addr Pointer to the 6-byte source MAC address, extracted by local methods from the
MAC header.

ether_type Ethernet-type field from the MAC header.

RETURN VALUES

Always returns 0.

CONSTRAINTS

None

SEE ALSO

set_up_8022(9F), set_up_ip(9F), set_up_link(9F)
NAME

set_up_ip - Sets up global information for the filter and formatting functions.

SYNOPSIS

#include <fmt.h>
#include <nt1.h>
#include <sys/subsys_id.h>

int set_up_ip(
    u_char *buf_ptr,
    int len
) ;

DESCRIPTION

The set_up_ip function sets up global information used by the filter_packet, format_link_nice, format_link_raw, and format_link_terse filter and formatting functions. It walks through the buffer and copies protocol header information to the appropriate global variables used by the filter and formatters.

NOTE: Most link products must not use this routine. Call this routine only when no link information is available for formatting output (for example, NS_LOOPBACK).

PARAMETERS

buf_ptr Pointer to the beginning of the IP information, which must not include MAC information.

len Length of the buffer, excluding the MAC header.

RETURN VALUES

Always returns 0.

CONSTRAINTS

None

SEE ALSO

set_up_8022(9F), set_up_ether(9F), set_up_link(9F)
set_up_link(9F)

NAME

set_up_link - Sets up global information for the link layer only.

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>

int set_up_link(
    u_char *buf_ptr,
    int len,
    u_char *dst_addr,
    u_char *src_addr
);

PARAMETERS

buf_ptr Pointer to beginning of the Data Link information. It must not include MAC
information. The routine does not currently use this parameter, but it is here for
future extensions.

len Length of the buffer, excluding the MAC header.

dst_addr Pointer to the 6-byte destination MAC address, extracted by local methods from
the MAC header.

src_addr Pointer to the 6-byte source MAC address, extracted by local methods from the
MAC header.

DESCRIPTION

The set_up_link function sets up global information only for the link layer and does not
attempt to extract any upper layer information from the traced packet. It does the minimum
setup necessary to use the format_link_nice, format_link_raw, and format_link_terse
functions.

NOTE: Use this routine only if the packet being formatted cannot be handled by set_up_8022.

RETURN VALUES

Always returns 0.

CONSTRAINTS

None

SEE ALSO

set_up_8022(9F), set_up_ether(9F), set_up_ip(9F)
ss_N_fmt_flag_type(9S)

NAME

ss_N_fmt_flag_type - Flags structure that control the behavior of formatter output.

SYNOPSIS

#include<fmt.h>

DESCRIPTION

The ss_N_fmt_flag_type structure contains flags that dictate the desired format of trace and log data output. This structure is exchanged between the core formatter and subformatter through various functions, such as tl_header_format1 and subsys_N_format.

Structure

The ss_N_fmt_flag_type structure has the following field names:

Table 4-3 Structure Fields (ss_N_fmt_flag_type)

<table>
<thead>
<tr>
<th>Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned</td>
<td>verbosity_bit: 1</td>
</tr>
<tr>
<td>unsigned</td>
<td>console_logging: 1</td>
</tr>
<tr>
<td>unsigned</td>
<td>highlight_bit: 1</td>
</tr>
<tr>
<td>unsigned</td>
<td>nice_mode_bit: 1</td>
</tr>
<tr>
<td>unsigned</td>
<td>terse_mode_bit: 1</td>
</tr>
<tr>
<td>unsigned</td>
<td>terse_link_mode_bit: 1</td>
</tr>
<tr>
<td>unsigned</td>
<td>terse_time_mode_bit: 1</td>
</tr>
<tr>
<td>unsigned</td>
<td>map_to_names_bit: 1</td>
</tr>
<tr>
<td></td>
<td>reserved: 23</td>
</tr>
</tbody>
</table>

**verbosity_bit**

When this bit is set, a high level of verbosity has been selected (high verbosity is the default).

**console_logging**

This bit is set if console logging is enabled, in which case the subformatter must only call the tl_header_format1 routine and provide very minimal additional information (to be kept to one line).

**highlight_bit**

This bit is set when nice formatting has been enabled (nice output not enabled is the default). Nice formatting is the most descriptive mode of formatting. All possible information must be displayed in this mode of output. Nice mode is not usually used for log messages.

**terse_mode_bit**

This bit is set when terse formatting has been enabled (terse output not enabled is the default). Terse formatting must be limited to one line of output per trace record. Terse mode is not usually used for log messages.

**terse_link_mode_bit**

If terse mode is enabled then terse_link_mode_bit is a flag that must cause the link name to be included in the output.

**terse_time_mode_bit**

If terse mode is enabled, terse_time_mode_bit is a flag that must cause the timestamp to be included in the output.
map_to_names_bit  This bit is set when numbers must be resolved into names whenever possible (mapping numbers to names is enabled by default). For example, an IP address must be displayed as a host name if the map_to_names_bit flag is set.

reserved  Reserved

SEE ALSO

tl_header_format1(9F), subsys_N_format(9F)
NAME

ss_N_fmtParms_type - Structure containing information required by subformatter.

SYNOPSIS

#include<fmt.h>

DESCRIPTION

The ss_N_fmt_Parms_type structure contains all the information required by a subformatter to operate. The subformatter can obtain this information by calling the tl_get_Parms function. The core formatter builds and initializes this data structure before calling subsys_N_format.

Structure

The ss_N_fmt_Parms_type structure has the following field names:

Table 4-4 Structure Fields (ss_N_fmt_Parms_type)

<table>
<thead>
<tr>
<th>Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *</td>
<td>ss_status_ptr</td>
</tr>
<tr>
<td>FILE *</td>
<td>ss_output_strm</td>
</tr>
<tr>
<td>int</td>
<td>ss_output_fd</td>
</tr>
<tr>
<td>FILE *</td>
<td>ss_error_strm</td>
</tr>
<tr>
<td>int</td>
<td>ss_error_fd</td>
</tr>
<tr>
<td>nl_catd</td>
<td>ss_msg_catd</td>
</tr>
<tr>
<td>char *</td>
<td>ss_name</td>
</tr>
<tr>
<td>char *</td>
<td>ss_binary_msg_ptr</td>
</tr>
<tr>
<td>char *</td>
<td>ss_options_ptr</td>
</tr>
<tr>
<td>ss_N_fmt_flag_type</td>
<td>ss_n_fmt_flag</td>
</tr>
<tr>
<td>char *</td>
<td>time_buffer</td>
</tr>
<tr>
<td>int</td>
<td>time_buffer_length</td>
</tr>
<tr>
<td>int</td>
<td>output_file_count</td>
</tr>
<tr>
<td>fd_result</td>
<td>output_files[1]</td>
</tr>
<tr>
<td>int</td>
<td>print_op</td>
</tr>
<tr>
<td>int</td>
<td>user_count</td>
</tr>
<tr>
<td>user_acct_result *</td>
<td>users</td>
</tr>
<tr>
<td>int</td>
<td>initied_flag</td>
</tr>
<tr>
<td>int</td>
<td>nettl_version</td>
</tr>
</tbody>
</table>

ss_status_ptr      Used by a subformatter to store an error code if it fails.
ss_output_strm     A FILE pointer that will receive the formatted trace/log message.
ss_output_fd       A file descriptor that will receive the formatted trace/log messages.
ss_error_strm      A FILE pointer that will receive any fatal or nonfatal error messages.
ss_error_fd        A file descriptor that will receive any fatal or nonfatal error messages.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss_msg_cat</td>
<td>The subsystems' message catalog descriptor to be used in catgets.</td>
</tr>
<tr>
<td>ss_name</td>
<td>A pointer to the subsystem name.</td>
</tr>
<tr>
<td>ss_binary_msg_ptr</td>
<td>A pointer to a buffer containing log/trace messages to be formatted.</td>
</tr>
<tr>
<td>ss_options_ptr</td>
<td>A pointer to a buffer containing information to be passed between the subsys_N_format routine and the subsys_N_get_options routine. See the options-ptr parameter in subsys_N_format(9F).</td>
</tr>
<tr>
<td>ss_n_fmt_flags</td>
<td>Options flags of type ss_N_fmt_flag_type, which indicates the desired format of output.</td>
</tr>
<tr>
<td>time_buffer</td>
<td>A string containing the formatted timestamp from the trace/log header (see the time-buffer parameter of subsys_N_format(9F)).</td>
</tr>
<tr>
<td>time_buffer_length</td>
<td>Length of the time_buffer string, not counting the null terminator.</td>
</tr>
<tr>
<td>output_file_count</td>
<td>The number of output files to receive the formatted trace/log messages. For HP-UX, this member must have a value of 1.</td>
</tr>
<tr>
<td>output_files[]</td>
<td>An array of structures consisting of a file descriptor and result variable for each file to receive the formatted trace/log output. For HP-UX, only output_file[0].fd refers to the file receiving the formatter output.</td>
</tr>
<tr>
<td>print_op</td>
<td>For HP-UX, this member has a value of 0.</td>
</tr>
<tr>
<td>user_count</td>
<td>For HP-UX, this member has a value of 0.</td>
</tr>
<tr>
<td>users</td>
<td>For HP-UX, this member has a value of NULL.</td>
</tr>
<tr>
<td>inited_flag</td>
<td>Flag which indicates if the structure is already (initialized or not.)</td>
</tr>
<tr>
<td>nettl_version</td>
<td>Gives the current nettl version.</td>
</tr>
</tbody>
</table>

**SEE ALSO**
catgets(9F), tl_get_parms(9F), ss_N_fmt_flag_type(9S), subsys_N_fomat(9F), subsys_N_get_options(9F)
NAME

subsys_get_cmd_preview - Request to get the CLI commands that will be used to execute the selected task.

SYNOPSIS

#include <sys/dlpi_ext.h>
#include <netmod.h>

int32_t subsys_get_cmd_preview(
    uint32_t *version_ptr,
    netmod_tab_code_t tab,
    void *svcs,
    const char *if_name,
    const netmod_task_info_t *task_info,
    char **cmd_preview,
    netmod_freefunc_t *freefunc,
    void *rsvd
);

PARAMETERS

version_ptr Version of the shared library that is compatible with the NIC tool or the version of the NIC tool that is compatible with the shared library, whichever is applicable.

tab Integer code for the tab area from which the library API is called.

svcs Shared library handle for nwmgr services returned by dlopen of the nwmgr services library.

if_name Name of the selected interface. This can be NULL for subsystem tasks.

task_info The NIC tool must modify the task qualifiers to the list that the user has chosen, and similarly for other qualifiers. The NIC tool must indicate with the CHANGED flag each attribute that has changed.

cmd_preview String containing a preview of the commands that will be executed for the selected task.

freefunc Function pointer that the NIC tool invokes with the pointer to subsystem tasks. This parameter is necessary if the information returned (if_tasks, if_details, task_info, or out_task, depending on the routine) is dynamically allocated by the NIC driver shared library. If freefunc is not NULL, it is called with the info pointer (depending on the routine).

rsvd Reserved parameter. Set to NULL in the first release.

DESCRIPTION

The subsys_get_cmd_preview routine returns the command Preview string for a given task (and interface, if any). The string contains a set of one or more commands, which when executed in that sequence results in the same action that is performed when OK is clicked on the task page. Implementation of this routine is optional for a subsystem; the command preview is required only if the library supports a task for which the NTI_FLAG_DELEGATE_PREVIEW flag is not set.

This routine does not need a catalog parameter because the command preview string is not localized. If the command preview string has to be localized, the string must contain the localized message provided by the subsystem.
The NIC tool invokes the `subsys_get_cmd_preview` routine when the user clicks the Preview button in the task page. The user might not have changed any field in the task page.

**RETURN VALUES**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>Negative integer</td>
<td>Failure. Negative of an error code from the <code>errno.h</code> header file. For example, –ENOMEM</td>
</tr>
</tbody>
</table>

**CONSTRAINTS**

This routine cannot be the first call to a subsystem library. This call must be preceded by a call to `subsys_get_task_info`.

The NIC tool routines and `nwmgr` routines do not allow subsystems to invoke `exit`.

**SEE ALSO**

`netmod_task_info_t(9S)`
subsys_get_if_details (9F)

NAME

subsys_get_if_details - Request to get details of the selected interface.

SYNOPSIS

#include <sys/dlpi_ext.h>
#include <netmod.h>

int32_t subsys_get_if_details(
    uint32_t *version_ptr,
    netmod_tab_code_t tab,
    void *svcs,
    netmod_group_t **if_details,
    char **catalog,
    netmod_freefunc_t *freefunc,
    void *rsvd
);

PARAMETERS

version_ptr Version of the shared library that is compatible with the NIC tool or the version of the NIC tool that is compatible with the shared library, whichever applicable.

tab Integer code for the tab area from which the library API is called.

svcs Shared library handle for nwmgr services returned by dlopen() of nwmgr services library.

if_name Name of the selected interface.

if_details List of attribute groups for the selected interface to be displayed by the NIC tool. The details are displayed in the Details of Interface area in the main page of the selected tab. The list may or may not be NULL-terminated. When the list is null-terminated, the terminating null character must not be included.

catalog Name of the message catalog file that contains the localization information for the subsystem library.

freefunc Function pointer that the NIC tool invokes with the pointer to subsystem tasks. This parameter is necessary if the information returned (if_tasks, if_details, task_info, or out_task, depending on the routine) is dynamically allocated by the NIC driver shared library. If the freefunc is not NULL, it is called with the info pointer (depending on the routine).

rsvd Reserved parameter. Set to NULL in the first release.

DESCRIPTION

The subsys_get_if_details routine returns the list of attribute groups that must be displayed in the Details of Interface for a selected interface in the main page of the tab. This routine returns the same data for a given context no matter how often it is called.

When the user selects an interface in any of the tab areas, the NIC tool invokes the corresponding subsystem’s routine.

RETURN VALUES

>=0 The number of attribute groups to be displayed. This number may be zero. The NIC tool must depend only on this number and not on the NULL termination of the list.

Negative integer Failure/Error
**CONSTRAINTS**

The NIC tool routines and nwmgr routines do not allow subsystems to invoke `exit`.

**SEE ALSO**

`netmod_tab_code_t(9S), netmod_group_t(9S)`
**NAME**

subsys_get_if_tasks - Request to get the list of tasks relevant to the selected interface.

**SYNOPSIS**

```c
#include <sys/dlpi_ext.h>
#include <netmod.h>

int32_t subsys_get_if_tasks(
    uint32_t *version_ptr,
    netmod_tab_code_t tab,
    void *svcs,
    netmod_string_t **if_tasks,
    char **catalog,
    netmod_freefunc_t *freefunc,
    void *rsvd
);
```

**PARAMETERS**

- **version_ptr**
  Version of the shared library that is compatible with the NIC tool or the version of the NIC tool that is compatible with the shared library, whichever applicable.

- **tab**
  Integer code for the tab area that the library API is called from.

- **svcs**
  Shared library handle for nwmgr services returned by dlopen of the nwmgr services library.

- **if_name**
  Name of the selected interface.

- **if_tasks**
  Link list of tasks that the selected interface supports.

- **catalog**
  Name of the message catalog file that contains the localization information for the subsystem library.

- **freefunc**
  Function pointer that the NIC tool invokes with the pointer to subsystem tasks. This parameter is necessary if the information returned (if_tasks, if_details, task_info, or out_task, depending on the routine) is dynamically allocated by the NIC driver shared library. If the freefunc is not NULL, it is called with the info pointer (depending on the routine).

- **rsvd**
  Reserved parameter. Set to NULL in the first release.

**DESCRIPTION**

The subsys_get_if_tasks API returns the list of tasks that the user can select only after selecting an interface. For example, after selecting an interface the user can select the task View/Edit Current Values. This API is idempotent, that is, it returns the same data for a given context no matter how often it is called.

The NIC tool invokes the subsys_get_if_tasks API when the user selects an interface in any of the tab areas.

**RETURN VALUES**

- **>=0**
  The number of supported interface tasks. This number might be zero. The NIC tool must depend only on this number and not on the NULL termination of the list.

- **Negative integer**
  Failure/Error
CONSTRAINTS
The NIC tool APIs and nwmgr routines do not allow subsystems to invoke exit.

SEE ALSO
netmod_tab_code_t(9S), netmod_string_t(9S)
subsys_get_task_info (9F)

NAME

subsys_get_task_info - Request to get the task information to be displayed on the task page.

SYNOPSIS

#include <sys/dlpi_ext.h>
#include <netmod.h>

int32_t subsys_get_task_info(
    uint32_t *version_ptr,
    netmod_tab_code_t tab,
    void *svcs,
    const char *if_name,
    int32_t task_code,
    netmod_task_info_t **task_info,
    char **catalog,
    netmod_freefunc_t *freefunc,
    void *rsvd
);

PARAMETERS

version_ptr Version of the shared library that is compatible with the NIC tool or the version of the NIC tool that is compatible with the shared library, whichever applicable.

tab Integer code for the tab area from which the library routine is called.

svcs Shared library handle for nwmgr services returned by dlopen() of nwmgr services library.

if_name Name of the selected interface. This might be NULL for subsystem tasks.

task_code Unique task code of the task selected by the user.

task_info Task information for the selected task.

catalog Name of the message catalog file that contains the localization information for the subsystem library.

freefunc Function pointer that the NIC tool invokes with the pointer to subsystem tasks. This parameter is necessary if the information returned (if_tasks, if_details, task_info, or out_task, depending on the routine) is dynamically allocated by the NIC driver shared library. If the freefunc is not NULL, then it is called with the info pointer (depending on the routine).

rsvd Reserved parameter. Set to NULL in the first release.

DESCRIPTION

The subsys_get_task_info routine returns the task information for the selected task.

Every subsystem must ensure that each task it supports has a unique task code. While common tasks have standardized task codes, subsystems might use private task codes as well, which may not be unique across subsystems. So the task codes may be unique only within a subsystem, not necessarily across subsystems.

The NIC tool invokes the subsys_get_task_info routine when the user selects a task, either subsystem-specific or interface-specific.

RETURN VALUES

0 Success
Negative integer Failure. Negative of an error code from the `errno.h` header file. For example –EINVAL

**CONSTRAINTS**

The NIC tool routines and `nwmgr` routines do not allow subsystems to invoke `exit`.

**Postconditions:** For every attribute with a valid value, the `na_old_value` field has the same value as the `na_value` field. If the attribute type involves a pointer, `na_old_value` might point at the same location as `na_value`.

**SEE ALSO**

`netmod_tab_code_t(9S), netmod_task_info_t(9S)`. 


NAME

subsys_refresh_task_info - Request to get the data for the fields that are updated by the user.

SYNOPSIS

#include <sys/dlpi_ext.h>
#include <netmod.h>

int32_t subsystem_refresh_task_info(
    uint32_t *version_ptr,
    netmod_tab_code_t tab,
    void *svcs,
    const char *if_name,
    netmod_task_info_t **task_info,
    const netmod_task_info_t *in_task,
    netmod_task_info_t **out_task,
    char **catalog,
    netmod_freefunc_t *freefunc,
    void *rsvd
);

PARAMETERS

version_ptr Version of the shared library that is compatible with the NIC tool or the version of the NIC tool that is compatible with the shared library, whichever applicable.
tab Integer code for the tab area from which the library API is called.
svcs Shared library handle for nwmgr services returned by dlopen of the nwmgr services library.
if_name Name of the selected interface. This may be NULL for subsystem tasks.
task_info The NIC tool must modify the task qualifiers to the list that the user has chosen, and similarly modify other qualifiers. The NIC tool must indicate with the CHANGED flag each attribute that has changed.
in_task The task_info parameter that was most recently provided by the library to the caller. This data structure (and linkages) is recreated by the NIC tool.
out_task The task_info parameter with the modifications, if any, that the library might have made as part of executing the task. The out_task must not contain any reference to memory or pointers used in in_task.
catalog Name of the message catalog file that contains the localization information for the subsystem library.
freefunc Function pointer that the NIC tool invokes with the pointer to subsystem tasks. This parameter is necessary if the information returned (if_tasks, if_details, task_info, or out_task, depending on the API) is dynamically allocated by the NIC driver shared library. If the freefunc is not NULL, it is called with the info pointer (depending on the routine).
rsvd Reserved parameter. Set to NULL in the first release.

DESCRIPTION

The subsystem_refresh_task_info routine is required only if the subsystem specifies NA_REFRESH_ON_CHANGE (Refresh bit) for any attribute in any task info it provides to the NIC tool. It is necessary to implement this API if the task page contains any attributes set with the Refresh bit.
The NIC tool invokes the `subsys_refresh_task_info` routine when the user modifies any attribute which is marked for Refresh on Change.

**RETURN VALUES**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>Negative integer</td>
<td>Failure. Negative of an error code from the <code>errno.h</code> header file. For example –EINVAL</td>
</tr>
</tbody>
</table>

**CONSTRAINTS**

The NIC tool routines and `nwmgr` routines do not allow subsystems to invoke `exit`.

**SEE ALSO**

`netmod_tab_code_t(9S)`, `netmod_task_info_t(9S)`
subsys_N_format(9F)

NAME

subsys_N_format - Routine to format a single trace or log message from subsystem N.

SYNOPSIS

#include <fmt.h>
#include <nt1.h>
#include <sys/subsys_id.h>

int subsys_N_format(
    ss_N_fmt_flag_type flags,
    char *binary-msg-ptr,
    char *options-ptr,
    int msg-cat-fd,
    int error-fd,
    int output-file-count,
    fp_result output-files[],
    char *time-buffer,
    int time-buffer-length,
    int print-op,
    int user-count,
    user-acct-result users[],
    err_num status
);

DESCRIPTION

The subsys_N_format network function is provided by the subsystem developer. It can have any unique name. Pass the name to Network Services by including it in a shared library that is specified in the nettlgen.conf configuration file. With this naming notification method, several subsystems can use the same subsys_N_format function.

At run time, the netfmt command loads the library and calls the routine whenever data from the subsystem is encountered.

The subsys_N_format network function formats a single trace or log message from the N subsystem. It may discard the message based on filter information in the options file defined by the subsys_N_get_options function associated with the subsystem.

The successful integration of all subformatters for all subsystems requires that all subformatters follow prescribed guidelines. All subsystems must call tl_header_format1 for each binary message that they will be formatting.

Only the tl_format_fprintf, tl_format_write, and tl_raw_format functions must be used to produce output. Only the file descriptors and file pointers provided by the formatter may be used. No other files may be opened, and no user input may be solicited. No signals must be masked. The subformatter must never call exit. The definitions of the various flags must be carefully followed. Subsystems must try to mirror the behavior of other subsystems as much as possible.

NOTE: Do not use printf or write to produce output.

For example, to write messages to standard output, you would use the tl_format_write function with the output-files[0].fd file descriptor. To write messages to standard error, you would use the error-fd file descriptor.
For simplicity and compatibility with anticipated future growth, use the following mechanism to access the information a subformatter requires. A subsystem can call the `tl_get_parms` function anywhere within the subformatter to get a pointer to all of the information that a subformatter might need. Use this `tl_get_parms` mechanism whenever possible, as explained further in `tl_getParms(9F)`.

All future parameter changes will be made through the `tl_get_parms` function. For backwards compatibility, the old parameter list remains the same; but ignore it in favor of using the information returned by `tl_get_parms`.

**PARAMETERS**

- **flags**
  Flags of type `ss_N_fmt_flag_type` that indicates the desired output format.

- **binary-msg-ptr**
  A pointer to a buffer that contains the binary trace or log message to be formatted. The buffer contains the trace/log header, a `tl_msg_hdr_type` structure (from the `<ntl.h>` file), followed by the trace/log data (from `ktrc_write` or `klogg_write`):

```c
typedef struct {
    unsigned short     hdr_len;
    short              subsystemid;
    int                device_id;
    tl_msg_flag_type   flags;
    set_of_32          kind;
    set_of_32          class;
    set_of_32          version;
    int                dropped_events;
    unsigned int       dropped_data;
    unsigned int       data_len;
    unsigned int       orig_data_len;
    struct timeval     time;
    int                invoke_id;
    int                path_id;
    unsigned short     log_instance;
    short              uid;
    unsigned int       connection_id;
}  tl_msg_hdr_type;
```

**NOTE:** For tracing, the data may be truncated by the `nettl` command facilities. Check the `tl_msg_hdr_type>data_len` field to find out how much data was captured.

- **options-ptr**
  A pointer to a data structure defined by the subsystem for communication between the `subsys_N_get_options` routine and the `subsys_N_format` routine. If no options are used, this pointer is NULL. The actual type of the structure pointed to by `options-ptr` is entirely up to the subsystem developer.

- **msg-cat-fd**
  The file descriptor of the subsystem message catalog configured in `nettlgen.conf`. The formatter opens subsystem message catalogs using `catopen`. The `tl_check_cat_version` function can be used to check that the version of the message catalog corresponds to the version of the subformatter. Subsystems must not open their own message catalog files.

- **error-fd**
  A file descriptor that identifies the file that receives any fatal or nonfatal error messages (typically associated with `stderr`).

- **output-file-count**
  The number of output files to receive the formatted trace/log messages. It must be 1 for HP-UX.
output-files[] An array of structures, each of which contains a file descriptor number, \texttt{fd}, and a result.

\begin{verbatim}
typedef struct {
    int fd;
    int result;
} fp_result;
\end{verbatim}

The \texttt{fd}-designated output file, \texttt{output-files[0].fd}, receives the formatted trace/log messages. Only one output file is used for HP-UX; \texttt{output-files[0].result} is ignored. This output file will have been opened by the formatter driver.

time-buffer A string containing the formatted time stamp from the trace/log header.

time-buffer-length Length of the time-buffer string, not counting the null terminator.

print-op For HP-UX, this parameter must be 0.

user-count For HP-UX, this parameter must be 0.

users For HP-UX, this parameter must be NULL.

status Contains an error code value if the routine returns -1, indicating an error condition. \texttt{<fmt.h>} gives a complete list of such error codes:

\begin{verbatim}
FMTERR_INV_FLAGS  Invalid flags parameter.
FMTERR_INV_BIN_MP Invalid binary-msg-ptr binary message pointer.
FMTERR_INV_OUT_FP Invalid output file pointer.
FMTERR_INV_MC_FP Invalid message catalog file pointer.
FMTERR_INV_TL_MSG Invalid trace/log message. The message is so corrupted that no formatting can be done.
FMTERR_SYS_ERROR An error has been returned from a system call.
\end{verbatim}

**RETURN VALUES**

0 Successful completion.

-1 Error. An appropriate error code is provided in the status field and an error message (if any) is given in the file named in \texttt{error-fd}.

**CONSTRAINTS**

None

**SEE ALSO**

\texttt{catopen(9F), exit(9F), klogg_write(9F), ktrc_write(9F), subsys_N_get_options(9F), tl_check_cat_version(9F), tl_format_fprintf(9F), tl_format_write(9F), tl_get_parms(9F), tl_header_format1(9F), tl_raw_format(9F)}
subsys_N_get_options(9F)

NAME

subsys_N_get_options - Routine to process options for subsystem N.

SYNOPSIS

#include <fmt.h>

int subsys_N_get_options(
    get_opt_parms_type *get-opt-parms-ptr
);

DESCRIPTION

The subsys_N_get_options network function is provided by the subsystem developer. It can have any unique name. Pass the name to Network Services by including it in a shared library that is specified in the nettlgen.conf configuration file. With this naming notification method, several subsystems can use the same subsys_N_get_options function. Note that if the subsystem provides both subsys_N_format and subsys_N_get_options functions, they must be contained in the same shared library specified in the nettlgen.conf configuration file.

The core formatter reads the filter command file, collects the lines specific to a subsystem, then edits and stores them into a temporary file. It then calls your subsys_N_get_options routine with a parameter set as a pointer to this temporary file (given by subsys_strm). The subsys_N_get_options routine must call the tl_get_line routine to extract one line at a time from this temporary file for processing. The lines are upshifted, and redundant white space is removed. It is the responsibility of the subsys_N_get_options routine to read the subsystem-specific options information from the filter command file and store any necessary information.

PARAMETERS

get-opt-parms-ptr Pointer to a get_opt_parms_type structure that describes the subsystem. See get_opt_parms_type(9F) for explanation of the structure fields.

RETURN VALUES

0 Successful completion.
-1 Error. An appropriate error code is provided in the status_ptr field and an error message (if any) is given in the file named in error-strm.

CONSTRAINTS

None

SEE ALSO

get_opt_parms_type(9F), malloc(3C), nettl(1M), subsys_N_format(9F), tl_get_line(9F)
NAME

subsystem_name_netmgr_main - Acts as the subsystem shared library interface to nwmgr

SYNOPSIS

#include <netmod.h>

int32_t subsystem_name_netmgr_main(
    netmgr_cmd_t cmd,
    int32_t version,
    void *netmgr_lib_hdl,
    int32_t logfd,
    void *rsvd
);

PARAMETERS

cmd
version
netmgr_lib_hdl
logfd
rsvd

Parsed command-line options.
Global version number currently supported by nwmgr.
Handle to the nwmgr services shared library opened with dlopen.
Log file descriptor.
Reserved parameter for future use.

DESCRIPTION

The function serves as the subsystem shared library interface to nwmgr. Every subsystem shared
library is expected to provide this interface so that the nwmgr core can pass the parsed
command-line options to it for further processing. Subsystems are expected to return to nwmgr
core at the end of their processing, so that all cleanup operations can be carried out before exit.

If the return value of this interface is non-zero, it indicates failure. If the return value is positive,
the equivalent value is set to errno, and the command exit code returns the value from the
interface.

RETURN VALUES

0 Success
Negative integer Error

CONSTRAINTS

None

SEE ALSO

subsystem_nwmgr(1M)
tl_banner_char(9F)

NAME

tl_banner_char - Gets the character used for tl_header_format1 banner printing.

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>

char tl_banner_char(
    unsigned int kind_class
);

PARAMETERS

kind_class The trace kind or log class of the message.

DESCRIPTION

The tl_banner_char function obtains the character to be used when printing a header banner
with the tl_header_format1 function. The character is based on the type of log class or trace
kind. This function helps to ensure consistent banners for all trace and log messages.

RETURN VALUES

The tl_banner_char always returns a character to be used by tl_header_format1.

CONSTRAINTS

None

SEE ALSO

tl_header_format1(9F)
tl_check_cat_version(9F)

NAME

tl_check_cat_version - Checks compatibility between subsystem message catalog and subsystem formatter library.

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>

int tl_check_cat_version(
    int msgcatfd,
    int setnum,
    int msgnum,
    char *expectedversion,
    FILE *errstream
);

PARAMETERS

msgcatfd A file descriptor of the message catalog which contains the version string.
setnum The set number in the message catalog.
msgnum The message number in the message catalog.
expectedversion The version string that the message catalog is expected to contain.
errstream A FILE pointer to a stream that will receive error messages.

DESCRIPTION

The tl_check_cat_version function checks that the subsystem message catalog has a compatible version with the subsystem formatter library. If not, an appropriate warning message is issued.

This function is (optionally) called by the subsys_N_format and subsys_N_get_options functions.

RETURN VALUES

0 Successful completion. The versions match.
-1 Error. The versions do not match or the file descriptor of the message catalog is invalid.

CONSTRAINTS

None

SEE ALSO

tl_header_format1(9F) subsys_N_format(9F), subsys_N_get_options(9F)
tl_format_fprintf(9F)

NAME

tl_format_fprintf - Converts, format, and print arguments to standard output.

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>

int tl_format_fprintf(
    FILE *stream,
    fmt_wrt_flag_type flags,
    error_num status_ptr,
    char *format,
    [ arg]...
);

PARAMETERS

stream One of the FILE streams contained in the ss_N_fmt_parms_type structure returned by tl_get_parms.

flags Controls the output behavior of the tl_format_fprintf routine. The value must be set before calling tl_format_fprintf.

typedef struct {
    unsigned highlight       : 1;
    unsigned wait_to_write   : 1;
    unsigned reserved        : 30;
} fmt_wrt_flag_type;

highlight Write the format data in inverse video.

wait_to_write Reserved for future use.

status_ptr Contains the error value if the routine returns a -1.

format The format character string contains two types of objects: plain characters that are copied to the output stream and conversion specifications. Each string results in fetching 0 or more arguments, arg. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are ignored.

arg Argument for the format character string.

DESCRIPTION

The tl_format_fprintf network function converts, formats, and prints its arguments under control of the format. This routine behaves like printf but must be used instead to give netfmt control over the formatted buffer.

RETURN VALUES

0 Successful completion.

-1 Error.

Fatal errors are reported through the return value and the status_ptr parameter. All error messages (as follows) are written to the file pointed to by the error_fd parameter of the subsys_N_format routine:

FMTERR_FORMAT_FPRINTF An error occurred in writing to the output stream.

FMTERR_INV_L_STR Invalid line pointer string.
FMTERR_INV_OUT_FD  Invalid output file descriptor.
FMTERR_SYS_ERROR   An error has been returned from a system call within the
                   tl_format_write routine.

CONSTRAINTS

None

SEE ALSO

subsys_N_format(9F), tl_format_write(9F), tl_get_parms(9F), tl_raw_format(9F)
NAME

tl_format_write - Writes a buffer to standard output.

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>

int tl_format_write(
    u_char *input_line_ptr,
    int input_line_byte_count,
    int error_fd,
    fmt_wrt_flag_type flags,
    fd_result output_files[],
    int print_op,
    int user_count,
    user_acct_result users[],
    error_num *status_ptr
);

PARAMETERS

input_line_ptr A character string that contains the message to be printed to the output files. The input_line_ptr parameter need not be null-terminated nor ended with a newline.

input_line_byte_count The byte count of the input_line_ptr message string.

error_fd A file descriptor pointing to a file to receive error messages from the tl_format_write routine.

flags Controls output behavior of the tl_format_write routine. The value must be set before calling tl_format_write.

typedef struct {
    unsigned int highlight : 1;
    unsigned int wait_to_write : 1;
    unsigned int reserved : 30;
} fmt_wrt_flag_type;

highlight Write the input_line_ptr data in inverse video.

wait_to_write Reserved for future use.

reserved Reserved for future use.

output_file_count This is the number of output files to receive the formatted trace/log header output. For HP-UX, only one output file is used, and the value is always 1.

output_files[] An array of structures consisting of a file descriptor and result variable for each file to receive the formatted trace/log header output. For HP-UX, only one output file is used; output_file[0].fd refers to the file receiving the formatter output.

print_op For HP-UX, this parameter must have a value of 0.

user_count For HP-UX, this parameter must have a value of 0.

users For HP-UX, this parameter must have a value of NULL.
status_ptr

*DESCRIPTION*

The `tl_format_write` network function writes a buffer to standard output, pointed to by `output_file[0].fd`. The buffer may be created by one or more calls to the `sprintf` C library function.

*RETURN VALUES*

0  Successful completion.
-1  Error.

Fatal errors are reported through the return value and the `status_ptr` parameter. All error messages (as follows) are written to the file pointed to by `error_fd`:

- `FMTERR_FORMAT_WRITE` An error has occurred in writing to an output file.
- `FMTERR_INV_L_STR` Invalid line pointer string.
- `FMTERR_INV_OUT_FD` Invalid output file descriptor.
- `FMTERR_SYS_ERROR` An error has been returned from a system call within the `tl_format_write` routine.

*CONSTRAINTS*

None

*SEE ALSO*

`tl_format_fprint(9F), tl_raw_format(9F)`
NAME

tl_get_line - Obtains a line from a filter command file.

SYNOPSIS

```c
int tl_get_line(
    FILE *commandstream,
    char *line,
    int linesize,
    char *origline,
    int lineno,
    FILE *errorstream
);
```

PARAMETERS

- `commandstream`: A FILE pointer that points to the temporary filter command file containing a single subsystem's filter commands; typically the `subsys_strm` field of the `get_optParms_type` parameter to `subsys_N_get_options`.
- `line`: The buffer where `tl_get_line` stores the filter command line.
- `linesize`: Size of line (no more than 2048).
- `origline`: The `tl_get_line` routine stores the original filter command line as it appeared in the filter command file in `origline`.
- `lineno`: The `tl_get_line` routine stores the line number of `origline` in `lineno` as it appeared in the filter command file.
- `errorstream`: A FILE pointer to a stream that will receive error messages.

DESCRIPTION

The `tl_get_line` network function obtains a line from a filter command file. The core formatter reads the filter command file, collects the lines specific to a subsystem, then edits and stores them into a temporary file. It then calls your `subsys_N_get_options` routine with a parameter set as a pointer to this temporary file. The `subsys_N_get_options` routine must call the `tl_get_line` routine to extract one line at a time from this temporary file for processing. The lines are upshifted, and redundant white space is removed. The `tl_get_line` function stores the "cleaned" filter command line in this buffer.

RETURN VALUES

- 2: Successful completion.
- 0: End of file.
- <0: Error.

CONSTRAINTS

None

SEE ALSO

- `subsys_N_get_options(9F)`
**tl_get_Parms(9F)**

**NAME**

tl_get_Parms - Returns a pointer to an ss_N_fmt_parms_type data structure.

**SYNOPSIS**

```c
#include <fmt.h>
#include <nt1.h>
#include <sys/subsys_id.h>

ss_N_fmt_parms_type *tl_get_Parms(void);
```

**DESCRIPTION**

The tl_get_Parms network function returns a pointer to a ss_N_fmt_parms_type data structure, containing members that a subsystem subformatter needs in order to operate. The core formatter builds and initializes this data structure before calling subsys_N_format. See ss_N_fmt_parms_type(9S) for description of this structure.

**RETURN VALUES**

- Successful completion. The value is a pointer to an ss_N_fmt_parms_type data structure containing members that a subsystem subformatter needs to operate.

**CONSTRAINTS**

None

**SEE ALSO**

subsys_N_format(9F)
NAME

tl_header_format1 - Routine to format a single trace or log header.

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>

int tl_header_format1(
    char *header_ptr,
    int error_fd,
    ss_N_fmt_flag_type flags,
    char *kind_str,
    char banner_char,
    int output_file_count,
    fd_result output_files[],
    char *time_buffer,
    int time_buffer_length,
    int print_op,
    int user_count,
    user_acct_result users[],
    int location,
    error_num *status_ptr
);

PARAMETERS

header_ptr Points to a buffer that contains the header of the trace/log message to be formatted.

error_fd A file descriptor that refers to the file that will receive any error messages.

flags Option flags in an ss_N_fmt_flag_type structure, defined in <fmt.h> and described in ss_N_fmt_flag_type(9S).

kind_str A text message (typically the result of the tl_log_class or tl_trace_kind function) to be displayed for the kind field from the trace/log header. This string must be null-terminated. The kind message is truncated to 16 characters. If kind_str is NULL, the kind field from the header is displayed as a decimal value.

banner_char The character to use in the banner header line (typically the result of the tl_banner_char function). The subformatter may use this character to indicate differences in messages, such as inbound or outbound messages. For example, inbound messages could use the character 'v' while outbound messages could use the character '^'.

output_file_count This is the number of output files to receive the formatted trace/log header output. For HP-UX, only one output file is used; this value is always 1.

output_files[] An array of structures consisting of a file descriptor and a result variable for each file to receive the formatted trace/log header output. For HP-UX, only one output file is used:
output_file[0].fd.
time_buffer

A string depicting the formatted time stamp from the trace/log header.

time_buffer_length

The length of time_buffer not counting the null terminator byte.

print_op

For HP-UX, this parameter must have a value of 0.

user_count

For HP-UX, this parameter must have a value of 0.

users

For HP-UX, this parameter must have a value of NULL.

location

A value that can be used to locate the source of the message in the code. This parameter is set by the subsystem and may be used to represent any information the subsystem desires.

status_ptr

The error value if the routine returns a -1:

- FMTERR_INV_HDR The trace/log header is invalid (corrupt).
- FMTERR_INV_HDR_PTR The trace/log header pointer is invalid.
- FMTERR_INV_OUT_FD The output file descriptor is invalid.
- FMTERR_INV_MC_FD The message catalog descriptor is invalid.
- FMTERR_SYS_ERROR An error was returned from a system call within tl_header_format1.

DESCRIPTION

The tl_header_format1 network function formats a single trace or log header. The format of the output conforms to the standard HP-UX network tracing and logging recommendations. The formatted header is written to the output file specified by output_file[0]. fd. The tl_header_format1 function must be called by every subformatter after the subsystem filters have been processed. At a minimum, this may be the only output generated by the subformatter.

RETURN VALUES

0 Successful completion.
-1 Error. Fatal errors are reported through the status_ptr parameter.

All error messages are written to the file pointed to by error_fd parameter.

CONSTRAINTS

None

SEE ALSO

tl_banner_char(9F), tl_get_parms(9F), tl_log_class(9F), tl_trace_kind(9F)
tl_log_class(9F)

NAME

tl_log_class - Returns a text interpretation for a log class value.

SYNOPSIS

char *tl_log_class(
    unsigned int class
);

PARAMETERS

class The numeric log class of the message. The keywords are defined in <sys/subsys_id.h>.

class Keyword
1    INFORMATIVE
2    WARNING
4    ERROR
8    DISASTER

DESCRIPTION

The tl_log_class network function returns a text interpretation of a log class. The log class is stored as an integer. This function converts that number into a string that can be used in the formatted output. For example, passing in a log class of 8 causes the return value to be DISASTER.

The result of tl_log_class is typically used as a parameter to tl_header_format1 when printing a header.

RETURN VALUES

<>NULL A pointer to a text interpretation of a log class.
NULL class is not a defined value.

CONSTRAINTS

None

SEE ALSO

tl_trace_kind(9F)
**NAME**

tl_raw_format - Formats trace or log message into hexadecimal and printable ASCII characters.

**SYNOPSIS**

```c
#include <fmt.h>
#include <ntl.h>
#include <sys/subsys_id.h>

int tl_raw_format(
    char *data_ptr,
    int num_bytes,
    int start,
    int error_fd,
    raw_fmt_flag_type flags,
    int output_file_count,
    fd_result output_files[],
    int print_op,
    int user_count,
    user_acct_result users[],
    error_num *status_ptr
);
```

**PARAMETERS**

- **data_ptr**
  A pointer to the buffer that contains the data to be dumped in hexadecimal form.

- **num_bytes**
  The number of bytes to dump from the buffer pointed to by `data_ptr`. There is no checking to ensure that the number of bytes given does not exceed the actual buffer length. If `num_bytes` is zero, no data will be dumped.

- **start**
  The offset into the buffer pointed to by `data_ptr` where the dump is to begin. If start is zero, the dump begins at the byte pointed to by `data_ptr`.

- **error_fd**
  A file descriptor that will receive error messages.

- **flags**
  Reserved for future use; must be set to 0 by the caller.

- **output_file_count**
  The number of output files to receive the raw dump. For HP-UX, this parameter must have a value of 1.

- **output_files**
  An array of structures, each of which contains a file descriptor and a result code for the last operation on the file. For HP-UX, only one output file is used; `output_files[0].fd` refers to the file receiving the formatter output.

- **print_op**
  For HP-UX, this parameter must have a value of 0.

- **user_count**
  For HP-UX, this parameter must have a value of 0.

- **users**
  For HP-UX, this parameter must have a value of NULL.

- **status_ptr**
  The error value if the routine returns -1.

**DESCRIPTION**

The `tl_raw_format` network function formats a trace or log message into both hexadecimal and printable ASCII characters. The raw formatted output appears as follows:
The left-most column gives the decimal byte offset. The center area is the hexadecimal display of the data. The right-most column is the printable ASCII display of the data. A period is displayed for any nonprinting character.

**RETURN VALUES**

0  Successful completion.

-1  Error. Fatal errors are reported through the `status_ptr` parameter.

All error messages are written to the file pointed to by the `error_fd` parameter.

**CONSTRAINTS**

None

**SEE ALSO**

`tl_format_fprintf(9F), tl_format_write(9F)`
tl_trace_kind(9F)

NAME

tl_trace_kind - Returns a text interpretation for a trace kind value.

SYNOPSIS

#include <sys/subsys_id.h>

char *tl_trace_kind(
    unsigned int kind
);

PARAMETERS

kind  The numeric trace kind of the message. The keywords are defined in <sys/subsys_id.h>.

<table>
<thead>
<tr>
<th>kind Value and Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000000 HDR_IN_BIT</td>
<td>Inbound Protocol Header.</td>
</tr>
<tr>
<td>0x40000000 HDR_OUT_BIT</td>
<td>Outbound Protocol Header.</td>
</tr>
<tr>
<td>0x20000000 PDU_IN_BIT</td>
<td>Inbound Protocol Data Unit (including header and data).</td>
</tr>
<tr>
<td>0x10000000 PDU_OUT_BIT</td>
<td>Outbound Protocol Data Unit (including header and data).</td>
</tr>
<tr>
<td>0x08000000 PROCEDURETRACE_BIT</td>
<td>Procedure entry and exit.</td>
</tr>
<tr>
<td>0x04000000 STATE_TRACE_BIT</td>
<td>Protocol or connection states.</td>
</tr>
<tr>
<td>0x02000000 ERROR_TRACE_BIT</td>
<td>Invalid events or condition.</td>
</tr>
<tr>
<td>0x01000000 LOGGING_TRACE_BIT</td>
<td>Special kind of trace that contains a log message.</td>
</tr>
<tr>
<td>0x00800000 LOOP_BACK_BIT</td>
<td>Packets whose source and destination system are the same.</td>
</tr>
<tr>
<td>0x00400000 PTOP_BIT</td>
<td>Packets whose transmission is point to point.</td>
</tr>
</tbody>
</table>

DESCRIPTION

The tl_trace_kind network function returns a text interpretation of a trace kind. The trace kind is stored as an integer. This function converts that number into a string that can be used in the formatted output. For example, passing in a trace kind of 0x80000000 causes the return value to be HDR IN TRACE. The result of tl_trace_kind is typically used as a parameter to tl_header_format1 when printing a header.

RETURN VALUES

<>NULL  A pointer to a text interpretation of a trace kind.

NULL  kind is not a defined value.

CONSTRAINTS

None

SEE ALSO

tl_log_class(9F)
NAME
tos_override_t - Types of TOS override for a VLAN.

SYNOPSIS
#include <sio/dlpi_ext.h>

DESCRIPTION
The types of TOS override for a VLAN are listed in this enumeration. Used by create and modify VLAN user-level requests and by drivers using the DL_PROP_VLAN_INFO property.

STRUCTURE MEMBERS
typedef enum pri_override {
    IN_IP_HEADER = 1,
    IN_ETHER_HEADER,
    IN_CONF_TOS,
    IN_CONF_PRI
} tos_override_t;

IN_IP_HEADER ToS in IP header will be used (default setting in DLPI).
IN_ETHER_HEADER Ethernet header 802.1p priority will be converted to ToS value.
IN_CONF_TOS User-specified ToS will be used.
IN_CONF_PRI User-specified 802.1p priority will be converted to ToS.

SEE ALSO
dl_hp_create_vlan_req_t(9S), dl_hp_modify_vlan_req_t(9S), pri_override_t(9S)
5 SCSI Reference Pages

SCSI Services is a set of commonly used SCSI functions that allow device and interface drivers to be much smaller and more supportable. In addition to providing most commonly used SCSI functions, WSIO SCSI Services also provides a supported pass-through mechanism.
**NAME**

dd_aen - SCSI driver entry point to manage asynchronous events.

**SYNOPSIS**

```c
void dd_aen(
    void *lpt_hdl,
    escsi_async_evt_t event,
    intptr_t ev_arg
);
```

**PARAMETERS**

- `lpt_hdl` Pointer to a lpt structure.
- `event` The asynchronous event being delivered. The following events are defined:
  - ESCSI_LPT_OFFLINE
  - ESCSI_LPT_ONLINE
- `ev_arg` Argument associated with the asynchronous event.

**DESCRIPTION**

This entry point allows the TP_OFFLINE/TP_ONLINE asynchronous notification from the interface driver to be delivered to the class driver. The class driver can take any specific action based on the event received. You must pass the dd_aen function pointer to the SCSI Services by specifying it in the dd_aen field of the escsi_ddsw_t structure.

**RETURN VALUE**

None

**CONSTRAINTS**

No locks are held when calling this routine.
This routine is called in the blocking context and can sleep.

**EXAMPLE**

```c
void mydriver_aen(
    void *lpt_hdl,
    escsi_async_evt_t event,
    intptr_t ev_arg)
{
    escsi_lpt_t *lpt = (escsi_lpt_t *)lpt_hdl;
    escsi_lun_t *lun = lpt->lun;

    switch (event) {
    case ESCSI_LPT_OFFLINE:
        /* Offline processing */
        escsi_lpt_offline(lpt);

        /* dequeue path from active path queue */
        ESCSI_LUN_LOCK(lun);
        if (escsi_is_on_q(&lun->active_path_q,
                         (escsi_list_t *)lpt) == ESCSI_TRUE) {
            escsi_deq_active_lpt(lun, lpt);
        }
        /* Perform class driver specific lunpath offline operation */
    }
/* Update lun->next_lpt with appropriate lun path*/

/* Compute path counts for the lun */
escsi_lun_compute_path_cnt(lun);
ESCSI_LUN_UNLOCK(lun);
break;

case ESCSI_LPT_ONLINE:
    /* Online processing */
escsi_lpt_online(lpt);
ESCSI_LUN_LOCK(lun);

    /* SCSI Services is responsible for the active path queue management for passthru luns */
    if (escsi_is_on_q(&lun->active_path_q, (escsi_list_t *)lpt) == ESCSI_FALSE) {
        escsi_enq_active_lpt(lun, lpt);
    }
    /* Perform class driver specific lunpath online operation */

    /* Update lun->next_lpt with appropriate lunpath*/

    /* Compute path counts for the LUN */
escsi_lun_compute_path_cnt(lun);
ESCSI_LUN_UNLOCK(lun);
break;

default:
    /*Unknown AEN*/
    break;

    /* Set the LUN health */
escsi_lun_set_health(lun);

        return;
    }

SEE ALSO

escsi_ddsw_t(9S), escsi_is_on_q(9F), escsi_lpt_offline(9F), escsi_lpt_online(9F),
escsi_lun_compute_path_cnt(9F), escsi_lun_set_health(9F), ESCSI_LUN_LOCK(9G)
dd_done(9E)

NAME

dd_done - SCSI driver entry point to handle post-I/O processing.

SYNOPSIS

void dd_done(
    struct buf *bp
);

PARAMETERS

bp    Pointer to a buf structure.

DESCRIPTION

The dd_done SCSI function is provided by the driver writer. It can have any unique name. The dd_done function pointer needs to be passed to SCSI Services by specifying it in dd_done field of escsi_ddsw_t structure. The purpose of this routine is to cleanup tasks necessary when an I/O completes.

RETURN VALUE

None

CONSTRAINTS

Called with the lunpath lock held.

EXAMPLE

void mydriver_done(struct buf *bp) {
{
    escsi_lpt_t *lpt = (escsi_lpt_t *) bp->B_LPT;
    escsi_lun_t *lun = lpt->lun;
    mydriver_lun_t *mydriver_lun = lun->dd_lun;
    /*Perform clean-up tasks necessary when this I/O is complete */
    return;
}

SEE ALSO

escsi_ddsw_t(9S)
**NAME**

dd_ioctl - SCSI driver entry point to allow or disallow ioctl commands sent through pass-through driver.

**SYNOPSIS**

```c
int dd_ioctl(
    dev_t dev,
    escsi_lun_t *lun,
    escsi_lpt_t *lpt,
    int cmd,
    caddr_t data,
    int flags
);
```

**PARAMETERS**

- **dev**  
  Device number.

- **lun**  
  Pointer to escsi_lun structure.

- **lpt**  
  Pointer to escsi_lpt structure.

- **cmd**  
  The ioctl command word.

- **data**  
  Pointer to the command parameters.

- **flags**  
  File access flags.

**DESCRIPTION**

The **dd_ioctl_okay** SCSI function is provided by the driver writer. It can be any unique name. The **dd_ioctl** function pointer needs to be passed to SCSI Services by specifying it in **dd_ioctl** field of the escsi_ddsw_t structure. If this routine exists in the escsi_ddsw_t structure, it is called by escsi_ioctl if the ioctl command remains unsatisfied by the choices provided within that SCSI Services procedure. If **dd_ioctl** does not exist when called, escsi_ioctl returns an error.

The following table shows the list of new ioctls supported in HP-UX 11i v3.

**Table 5-1 Ioctls Supported in HP-UX 11i v3**

<table>
<thead>
<tr>
<th>IOCTL</th>
<th>Device File</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIOC_DESCRIBE_EXT</td>
<td>New DSF</td>
</tr>
<tr>
<td>DIOC_RST_CLR</td>
<td>New DSF, Legacy DSF</td>
</tr>
<tr>
<td>SIOC_FORMAT/DIOC_FORMAT</td>
<td>New DSF, Legacy DSF</td>
</tr>
<tr>
<td>SIOC_DISK_EJECT</td>
<td>New DSF, Legacy DSF</td>
</tr>
<tr>
<td>SIOC_ERASE</td>
<td>New DSF, Legacy DSF</td>
</tr>
<tr>
<td>DIOC_EXCLUSIVE</td>
<td>New DSF, Legacy DSF</td>
</tr>
<tr>
<td>DIOC_BLKLIST_REMAP</td>
<td>New DSF, Legacy DSF</td>
</tr>
<tr>
<td>SIOC_VERIFY_BLANK</td>
<td>New DSF, Legacy DSF</td>
</tr>
</tbody>
</table>

**RETURN VALUE**

- **0**  
  Successful completion.

- **<0**  
  Error. The value is expected to be an error value.
EXAMPLE

```c
int mydriver_ioctl (dev_t dev,
    escsi_lun_t *lun,
    escsi_lpt_t *lpt,
    int cmd,
    caddr_t data,
    int flags)
{
    mydriver_lun_t      *mydriver_lun = lun->dd_lun;
    .
    .
    switch(cmd)
    {
        case SIOC_xxx:
            /*driver specific ioctl processing */
            error = mydriver_<drv_op>();
            break;
        default:
            error = EINVAL;
            break;
    }
    return error;
}
```

SEE ALSO

`escsi_ddsw_t(9S)`
dd_ioctl_okay(9E)

NAME

dd_ioctl_okay - SCSI driver entry point to allow/disallow ioctl commands sent through pass-through driver.

SYNOPSIS

```c
escsi_bool_t dd_ioctl_okay(
    dev_t dev,
    escsi_lpt_t *lpt,
    int cmd,
    caddr_t data,
    int flags
);
```

PARAMETERS

- `dev` Device number.
- `lpt` Pointer to `escsi_lpt` structure.
- `cmd` Command word.
- `data` Pointer to the command parameters.
- `flags` File access flags.

DESCRIPTION

The `dd_ioctl_okay` SCSI function is provided by the driver writer. It can be any unique name. The `dd_ioctl_okay` function pointer needs to be passed to SCSI Services by specifying it in `dd_ioctl_okay` field of the `escsi_ddsw_t` structure. By providing this function, device driver is given control over which ioctls are executed by the pass-through driver while they have a device open. This entry point is optional. If omitted, it implies the class driver permits all ioctls.

RETURN VALUE

- **ESCSI_TRUE** When pass-through ioctl can be issued.
- **ESCSI_FALSE** When pass-through ioctl must not be issued.

CONSTRAINTS

None

EXAMPLE

```c
escsi_bool_t mydriver_ioctl_okay(dev_t dev,
    escsi_lpt_t *lpt,
    int cmd,
    caddr_t data,
    int flags)
{
    /*Class driver specific ioctl processing */
    return ESCSI_TRUE;
}
```

SEE ALSO

`escsi_ddsw_t`(9S)
NAME
dd_io_init -- SCSI driver entry point to initialize the scb.

SYNOPSIS
void est_io_init(
    struct buf *bp
);

PARAMETERS
bp   Pointer to a buf structure.

DESCRIPTION
The dd_io_init SCSI function is provided by the driver writer. It can have any unique name. The dd_io_init function pointer needs to be passed to SCSI Services by specifying it in the dd_io_init field of escsi_ddsw_t structure. The purpose of this routine is to initialize the scb from the information in the bp.

RETURN VALUE
None

CONSTRAINTS
This function is called with lunpath lock held.

EXAMPLE
void mydriver_io_init(escsi_lpt_t *lpt,
    struct buf *bp,
    escsi_scb_t *scb)
{
    escsi_lun_t      *lun = lpt->lun;
    escsi_ulp_scb_t  *ulp_scb = (escsi_ulp_scb_t *)scb->ulp_scb;
    mydriver_lun_t   *mydriver_lun = lun->dd_lun;
    /* Initialize scb from the information in the bp. */
    ulp_scb->retry_cnt = <retry_cnt>;
    ulp_scb->sa  = <?????
    ulp_scb->max_retries = <max_retries>;
    return;
}

SEE ALSO
escsi_ddsw_t(9S)
**dd_lpt_init(9E)**

**NAME**

dd_lpt_init - SCSI driver entry point to allocate/initialize class driver specific resources.

**SYNOPSIS**

```c
void dd_lpt_init(
    escsi_lpt_t *lpt
);
```

**PARAMETERS**

*escsi_lpt_t* Pointer to a lpt structure.

**DESCRIPTION**

This entry point is called from SCSI Services in the context of discovery of new lunpaths. This routine is called to allow the class driver to initialize or allocate any class driver specific resources for the lunpath. It can be any unique name. The `dd_lpt_init` function pointer needs to be passed to the SCSI Services by specifying it in the `dd_lpt_init` field of `escsi_ddsw_t` structure.

**RETURN VALUE**

None

**CONSTRAINTS**

SCSI Services holds the lun semaphore around call to the `dd_lpt_init` entry point.

**EXAMPLE**

```c
void mydriver_lpt_init(escsi_lpt_t *lpt)
{
    mydriver_lpt_t *my_lpt = lpt->dd_lpt;

    /* LUN semaphore is held by calling SCSI Services */

    /*
     * The newly found lun-path must be added to lun->path_q.
     * Call the SCSI Services provided routine to do it.
     */
    ESCSI_LUN_LOCK(lpt->lun);
    escsi_new_lpt(lpt->lun);
    ESCSI_LUN_UNLOCK(lpt->lun);

    /* Initialize class driver lunpath (if any) */
    .
    .
    return;
}
```

**SEE ALSO**

`dd_lpt_uninit(9E)`, `escsi_ddsw_t(9S)`, `escsi_lpt_uninit(9F)`, `ESCSI_LUN_LOCK(9F)`, `escsi_new_lpt(9F)`
NAME

dd_lpt_uninit - SCSI driver entry point to uninitialize or deallocate class driver resources.

SYNOPSIS

void dd_lpt_uninit(
    escsi_lpt_t *lpt
);

PARAMETERS

lpt Pointer to a lpt structure.

DESCRIPTION

This entry point is called by SCSI Services whenever a lunpath is unregistered. This typically happens during one of the following operations:

- A PCI online deletion (OLD) of an interface card
- A dynamic kernel module (DLKM) unload of the underlying interface driver
- An rmsf -H hardware_path command performed on a lunpath hardware path or a target path hardware path.

The class driver can uninitialize or free any class driver specific resources for the lunpath.

RETURN VALUE

None

CONSTRAINTS

SCSI Services holds the lun semaphore around the call to the dd_lpt_uninit entry point.

EXAMPLE

void mydriver_lpt_uninit(escsi_lpt_t *lpt)
{
    /* LUN semaphore is held by calling SCSI Services */
    /* Free any class driver specific lpt resources */
    .
    .
    /* Uninitialize common lunpath resources */
    escsi_lpt_uninit(lpt);
    return
}

SEE ALSO

dd_lpt_init(9E), escsi_lpt_uninit(9F)
NAME

dd_lun_init - SCSI driver entry point to allow the device driver to allocate and initialize any additional resources for the lun.

SYNOPSIS

void dd_lun_init(
    escsi_lun_t *lun,
    escsi_lpt_t *lpt
);

PARAMETERS

escsi_lun_t Pointer to a lun structure.
escsi_lpt_t Pointer to a lpt structure.

DESCRIPTION

This entry point is called from SCSI Services in the context of discovery of new lun. This routine is called to allow the class driver to allocate and initialize any additional resources for that lun, and to load the class driver specific attributes for the lun, if any. The dd_lun_init function pointer needs to be passed to the SCSI Services by specifying it in the dd_lun_init field of the escsi_ddsw_t structure.

RETURN VALUE

None

EXAMPLE

void mydriver_lun_init(escsi_lun_t *lun,
    escsi_lpt_t *lpt)
{
    /*Device-specific lun initialization */
}

CONSTRAINTS

No locks are held when calling this routine.
The lun semaphore is held around this call to prevent parallel calls while the lun is being probed.
This routine is called in the blocking context and can sleep.

SEE ALSO

escsi_ddsw_t(9S)
DD_OPEN_CNT(9G)

NAME

DD_OPEN_CNT - Macro to return the number of pending class driver opens on a LUN device file.

SYNOPSIS

DD_OPEN_CNT(__lun)

PARAMETERS

__lun Pointer to a LUN structure.

DESCRIPTION

The DD_OPEN_CNT macro returns the number of opens performed on a LUN by the class driver claiming the LUN. It does not take into account the opens performed on a LUN by an esctl pass-through class driver.

RETURN VALUE

None

CONSTRAINTS

None

EXAMPLE

int mydriver_psize1(dev_t dev, int64_t *size) {
    escsi_lun_t *lun;
    mydriver_lun_t my_driver_lun;
    int_64_t nblks, rshift;

    *size = 0;

    // d_psize is a block driver entry point
    lun = escsi_devt_to_lun(dev, D_BLK);
    if (lun == NULL) {
        return(-1);
    }

    // Fail if the lun is not open
    if (DD_OPEN_CNT(lun) == 0) {
        return(-1);
    }
    .
    .
}

SEE ALSO

escsi_devt_to_lun(9F)
dd_pass_thru_done(9E)

NAME
dd_pass_thru_done  -  SCSI driver entry point to handle post pass-through I/O processing.

SYNOPSIS

void dd_pass_thru_done(
    struct buf *bp
);

PARAMETERS

bp    Pointer to a buf structure.

DESCRIPTION

The dd_pass_thru_done SCSI function is provided by the driver writer. It can be any unique name. The dd_pass_thru_done function pointer needs to be passed to SCSI Services by specifying it in dd_pass_thru_done field of the escsi_ddsw_t structure. The purpose of this function is to inform the class driver of the completion of pass-through I/O operation.

RETURN VALUE

None

CONSTRAINTS

None

EXAMPLE

mydriver_pass_thru_done(struct buf *bp){
{
    escsi_scb_t        *scb = (escsi_scb_t *)bp->B_SCB;
    escsi_lpt_t        *lpt = (escsi_lpt_t *)bp->B_LPT;
    escsi_lun_t        *lun = lpt->lun;
    mydriver_lun_t     *mydriver_lun = lun->dd_lun;
    u_char              *cdb = scb->cdb;
    sense_data_t       *sense_data = (sense_data_t *)scb->sense_data;
    /* Class driver specific operations based on the sense_data */
    return;
}

SEE ALSO

escsi_ddsw_t(9S)
**dd_pass_thru_okay(9E)**

**NAME**

dd_pass_thru_okay - SCSI driver entry point to control pass-through I/O requests.

**SYNOPSIS**

```c
escsi_bool_t dd_pass_thru_okay(
    escsi_cdbinfo_t *cdbinfo
);
```

**PARAMETERS**

- `cdbinfo` Pointer to an `escsi_cdbinfo_t` structure.

**DESCRIPTION**

The `dd_pass_thru_okay` SCSI function is provided by the driver writer. It can be any unique name. The `dd_pass_thru_okay` function pointer needs to be passed to SCSI Services by specifying it in `dd_pass_thru_okay` field of the `escsi_ddsw_t` structure. This is an optional field. Class driver is expected to provide this function, if it needs to control pass-through I/Os and cmdx I/Os that can be issued on the LUN. Absence of this function enables pass-through I/Os and cmdx I/Os unconditionally.

**RETURN VALUE**

- `ESCSI_TRUE`
- `ESCSI_FALSE`

**CONSTRAINTS**

None

**EXAMPLE**

```c
escsi_bool_t mydriver_pass_thru_okay(escsi_cdbinfo_t *cdbinfo)
{
    escsi_lun_t *lun = cdbinfo->lun;
    mydriver_lun_t *mydriver_lun = lun->dd_lun;
    u_char *cdb = cdbinfo->esioc_io->cdb;
    switch (cdb[0])
    {
        /*class driver specific switch/case functionality*/
    }
    return ESCSI_TRUE;
}
```

**SEE ALSO**

`escsi_ddsw_t(9S)`
NAME
dd_start - SCSI driver entry point to start an I/O request.

SYNOPSIS
struct buf *dd_start(
    escsi_lun_t *lun
);

PARAMETERS
lun Pointer to a lun structure.

DESCRIPTION
The dd_start SCSI function is provided by the driver writer. It can have any unique name. The
dd_start function pointer needs to be passed to SCSI Services by specifying it in the dd_start
field of the escsi_ddsw_t structure. This entry point is mandatory for all the class drivers and
the pass through driver.

RETURN VALUE
<> NULL Successful completion. A pointer to a valid buf structure is returned.
NULL Error.

EXAMPLE
typedef struct {
    .
    .
    struct buf *io_q;
    .
    .
} mydriver_lun_t

struct buf *
dd_start {
    escsi_lun_t *lun;
    mydriver_lun_t *mylun = lun->dd_lun;
    struct buf *bp;

    ESCSI_DEQ_BP_FROM_HEAD(&mylun->io_q, bp);
    return bp;
};

CONSTRAINTS
This function is called with LUN lock held.
This function cannot sleep.

SEE ALSO
escsi_ddsw_t(9S), ESCSI_DEQ_BP_FROM_HEAD(9M)
dd_strategy(9E)

NAME

dd_strategy - SCSI driver entry point to handle buf requests.

SYNOPSIS

void dd_strategy(
    escsi_lun_t *lun,
    struct buf *bp
);

PARAMETERS

lun    Pointer to a lun structure.
bp     Pointer to a buf structure.

DESCRIPTION

dd_strategy SCSI function is provided by the driver writer. It can have any unique name. The
dd_strategy function pointer needs to be passed to the SCSI Services by specifying it in the
dd_strategy field of the escsi_ddsw_t structure. This entry point is mandatory for all class
drivers except the pass-through driver.

RETURN VALUE

0    Successful completion
-1   I/O failed back to the upper layers because of an error.

CONSTRAINTS

The dd_strategy function must exist (be defined as non-NULL in the escsi_ddsw_t structure)
if the driver calls escsi_strategy. The dd_strategy function must not try to acquire SCSI
LUN lock as escsi_strategy calls dd_strategy while holding the LUN lock.

EXAMPLE

typedef struct {
    
    struct buf   *io_q;
    
} mydriver_lun_t

mydriver_strategy(escsi_lun_t *lun, struct buf *bp)
{
    mydriver_lun_t      *my_lun = lun->dd_lun;
    if (class driver specific error checking fails) {
        bp->b_flags    |= B_ERROR;
        bp->b_error    = ENXIO;
        bp->b_eei      = EEI_<eei>;
        bp->B_SCB       = NULL;
        bp->B_LPT       = NULL;
        ESCSI_LUN_UNLOCK(lun);
        escsi_iiodone(bp);
        ESCSI_LUN_LOCK(lun);
        Return -1;
    }
    /* Class driver specific processing on the I/O */
    
}
/ * Enqueue I/O to the class driver queue */
  ESCSI_ENQ_BP_AT_TAIL(&mylun->io_q, bp);
  return 0;
}

SEE ALSO
ESCSI_ENQ_BP_AT_TAIL(9M), escsi_ddsw_t(9S), escsi_iodeone(9F)
driver_activate_lpt(9F)

NAME

driver_activate_lpt - Active/passive (APSW) device plug-in function to activate a standby lunpath
<sys/escsi_services.h>

void driver_activate_lpt(
    escsi_cdbinfo_t *cdbinfo
);

PARAMETERS

cdbinfo Pointer to a escsi_cdbinfo_t data structure.

DESCRIPTION

The driver_activate_lpt function is the plug-in function to activate a standby lunpath of an active/passive array controller. This function must be registered by the kernel module that handles a specific set of active/passive array controllers. This function is mandatory.

The following fields of the escsi_cdbinfo_t data structure are relevant for this function:

cbfn Specifies an escsi_cbfn_t data structure that contains the pointer to the SCSI Services callback function (cbfn.cb_func) and the argument containing the SCSI Services handle (cbfn.cb_arg) to be passed to the callback function along with the result of the lunpath activation. The callback function must be called as follows:

• Failure to activate the lunpath:
  cdbinfo->cbfn.cb_func(cdbinfo->cbfn.cb_arg, EINVAL);
• Activation of lunpath successful:
  cdbinfo->cbfn.cb_func(cdbinfo->cbfn.cb_arg, ESUCCESS);

The following fields are important if the plug-in wants to reuse cdbinfo to send the I/O. The cdbinfo is also used by the caller of driver_activate_lpt (SCSI Services) for probing the lunpath. It is essential for proper functioning that driver_activate_lpt does not modify any fields in the cdbinfo except those required to issue the CDB. The cdbinfo structure is also accompanied by a 512-byte data buffer that can be used to accommodate any data transfer required for the CDB, provided the data transfer size is less than 512. The following fields can be modified:

escsio_io->flags Specifies the nature of the I/O. The following bit flags can be OR’ed:

- ESCTL_IO_LPT Indicates that this is a path-specific I/O. In his case, escsio_io->lpt_hwp must contain the hardware path of the lunpath on which to issue the I/O.

- ESCTL_READ Indicates a read-type I/O.

escsio_io->cdb Must be populated with the CDB (Command Descriptor Block) of the command being issued.

escsio_io->cdb_length Must specify the size of the CDB, in bytes.

escsio_io->data_xfer Must specify the maximum length, in bytes, of the data expected. It must not exceed 512. The data is returned in escsio_io->data.

escsio_io->max_msecs Specifies the I/O timeout value, ESCSI_IO_DFLT_TIMEOUT if the default timeout value is sufficient, or ESCSI_IO_INFINITE_TIME if the I/O must not be timed.
retries

Specifies the maximum number of times the I/O is to be retried, ESCSI_IO_DFLT_RETRIES for default number of retries, or 0 if the I/O must not be retried.

cbfn

Must specify the callback function and its argument. The callback function is called by SCSI Services to notify the completion of the I/O.

IMPORTANT: Before using this field, you must save its contents in the cbfn_ws field. This is necessary to be able to call the SCSI Services callback function to notify the status of the lunpath upon processing of the status of the I/O.

cbfn_ws

Used to save the contents of the cbfn field.

RETURN VALUE

None

CONSTRAINTS

The function is nonblocking; it cannot sleep.

This function is asynchronous. It is necessary to send a command to activate the lunpath. The function must return immediately after specifying a callback function to which SCSI Services returns after the I/O completion.

SEE ALSO

escsi_apsw_reg(9F), escsi_cdbinfo_t(9S), escsi_cmdx_ext(9F), escsi_get_inq_serial(9F)
**NAME**

`driver_check_lpt` - Active/passive (APSW) device plug-in function to determine the state of a lunpath

`<sys/escsi_services.h>`

```c
void driver_check_lpt(
    escsi_cdbinfo_t *cdbinfo
);
```

**PARAMETERS**

cdbinfo Pointer to an `escsi_cdbinfo_t` data structure.

**DESCRIPTION**

The `driver_check_lpt` function is the plug-in function to determine the active or standby status of a lunpath of an active/passive array controller. This function must be registered by the kernel module that handles a specific set of active/passive array controllers. This function is mandatory.

The following fields of the `escsi_cdbinfo_t` data structure are relevant for this function:

cbfn Specifies an `escsi_cbfn_t` data structure that contains the pointer to the SCSI Services callback function (`cbfn.cb_func`) and the argument containing the SCSI Services handle (`cbfn.cb_arg`) to be passed to the callback function along with the status of the lunpath. The callback function must be called as follows:

- Failure to determine the status of the lunpath:
  ```c
  cbdinfo->cbfn.cb_func(cbdinfo->cbfn.cb_arg, EINVAL);
  ```
- The lunpath is active:
  ```c
  cbdinfo->cbfn.cb_func(cbdinfo->cbfn.cb_arg, ESUCCESS);
  ```
- The lunpath is standby:
  ```c
  cbdinfo->cbfn.cb_func(cbdinfo->cbfn.cb_arg, ENOTCONN);
  ```

The following fields are important if the status of the lunpath can be determined by examining the inquiry data:

lpt_addr Data structure containing the address of the lunpath in the form of controller instance number (`lpt_addr.c`), target path instance number (`lpt_addr.t`), and lunpath instance number (`lpt_addr.l`). This field is important when the status of the lunpath can be determined by examining the inquiry data. The caller can use the `lpt_addr` field to initialize the `escsi_inq_serial_t` data structure passed to the `escsi_get_inq_serial` function to get the inquiry data stored in the lunpath data structure maintained by SCSI Services.

The following fields are important if the plug-in must send an I/O to the device to determine the status of the lunpath and the plug-in wants to reuse `cdbinfo` to send the I/O. The `cdbinfo` is also used by the caller of `driver_check_lpt` (SCSI Services) for probing the lunpath. It is essential for proper functioning that `driver_check_lpt` does not modify any fields in the `cdbinfo` except those required to issue the CDB. The `cdbinfo` structure is also accompanied by a 512-byte data buffer that can be used to accommodate any data transfer required for the CDB, provided the data transfer size is less than 512. The following fields can be modified:

escsio_io->flags Specifies the nature of the I/O. The following bit flags can be OR'ed:

- `ESCTL_IO_LPT` Indicates that this is a path-specific I/O. In this case, `escsio_io->lpt_hwp` must...
contain the hardware path of the lunpath on which to issue the I/O.

**ESCTL_READ** Indicates a read-type I/O.

- **escsio_io->cdb** Must be populated with the CDB (Command Descriptor Block) of the command being issued.
- **escsio_io->cdb_length** Must specify the size of the CDB, in bytes.
- **escsio_io->data_xfer** Must specify the maximum length, in bytes, of the data expected. It must not exceed 512. The data is returned in **escsio_io->data**.
- **escsio_io->max_msecs** Specifies the I/O timeout value, **ESCSI_IO_DFLT_TIMEOUT** if the default timeout value is sufficient, or **ESCSI_IO_INFINITE_TIME** if the I/O must not be timed.
- **retries** Specifies the maximum number of times the I/O is to be retried, **ESCSI_IO_DFLT_RETRIES** for default number of retries, or 0 if the I/O must not be retried.
- **cbfn** Must specify the callback function and its argument. The callback function is called by SCSI Services to notify the completion of the I/O.

**IMPORTANT:** Before using this field, you must save its contents in the **cbfn_ws** field. This is necessary to be able to call the SCSI Services callback function to notify the status of the lunpath upon processing of the status of the I/O.

- **cbfn_ws** Used to save the contents of the **cbfn** field.

**RETURN VALUE**

None

**CONSTRAINTS**

The function is nonblocking; it cannot sleep.

Can be asynchronous when the function needs to wait for I/O completion. In this case, the function must return immediately after specifying a callback function to which SCSI Services returns after the I/O completion.

Can be synchronous. It can return after notifying SCSI Services the state of the lunpath if there is no need to issue an I/O to the device to determine the state (for example, if the state can be determined by examining inquiry data).

**SEE ALSO**

escsi_apsw_reg(9F), escsi_cdbinfo_t(9S), escsi_cmdx_ext(9F), escsi_get_inq_serial(9F)
NAME

driver_inspect_lpt_err - Active/passive (APSW) device plug-in function to process check conditions and device-specific sense key data.

<sys/escsi_services.h>

void driver_inspect_lpt_err(
    escsi_addr_t *lpt_addr,
    escsi_scb_t *scb,
    intptr_t *error,
    intptr_t *msecs
);

PARAMETERS

lpt_addr Pointer to the lunpath address information (escsi_addr_t).
scb Pointer to the escsi_scb_t data structure containing information about the I/O, and its status and sense data.
error Pointer to error data.
msecs Indicates the length of time (in msecs) to wait before retrying the I/O. -1 means do not retry the I/O.

DESCRIPTION

The driver_inspect_lpt_err function is the plug-in function that is called in the context of I/O completion processing each time an I/O operation completes on a lunpath with a check condition. The function can determine the following:

- Whether or not to retry the I/O.
- If the I/O it to be retried, the amount of time to wait.
- Whether the lunpath has gone offline or is in a standby status.

This function must be registered by a kernel module that handles a specific set of active/passive array controllers. This function is optional.

RETURN VALUE

ENOTCONN The lunpath is in a standby state.
ENXIO The lunpath is offline.
ESUCCESS The lunpath is in an active state.

CONSTRAINTS

The function is nonblocking; it cannot sleep.
Synchronous.
Must not modify the sense data in the scb data structure.

SEE ALSO

escsi_apsw_reg(9F), escsi_cdbinfo_t(9S), escsi_cmdx_ext(9F), escsi_get_inq_serial(9F)
NAME

driver_ioctl - Mass storage interfaces driver ioctl entry point

SYNOPSIS

#include <sys/conf.h>

int driver_ioctl(
    dev_t dev,
    int cmd,
    caddr_t arg_ptr,
    int flag
);

PARAMETERS

dev    Device number.

cmd    Command word.

arg_ptr Pointer to the command word arguments, if any.

flag   File access flags.

DESCRIPTION

The driver_ioctl WSIO function is provided by the driver writer. It can have any unique
name. Pass the name to WSIO Services by specifying it in the d_ioctl field of the drv_ops
structure. Commonly, driver is replaced by the driver's name.

The driver_ioctl function supports the following ioctl commands for mass storage devices:

IOC_EFI2STR _IOWR('C', 9, struct ipf_efi2str_data)
    Converts a device path and returns a display string in the ipf_efi2str_data
    structure.

IOC_EFI2HW _IOWR('C', 10, struct ipf_efi2hw_data)
    Converts a device path and returns a hardware path in the ipf_efi2hw_data
    structure.

IOC_HW2EFI _IOWR('C', 11, struct ipf_hw2efi_data)
    Converts a hardware path and returns a device path in the ipf_hw2efi_data
    structure.

SEE ALSO

ipf_efi2str_data(9S), ipf_efi2hw_data(9S), ipf_hw2efi_data(9S)
NAME
escsi_action - Service to perform common I/O completion processing tasks.

SYNOPSIS

```c
int escsi_action(
    struct buf *bp,
    intptr_t flags,
    intptr_t errno,
    intptr_t eei,
    intptr_t msecs
);
```

PARAMETERS

- `bp` Pointer to a `buf` structure.
- `flags` Indicates if I/O is to be retried or not.
- `errno` I/O completion status.
- `eei` I/O completion status.
- `msecs` Retry delay time.

DESCRIPTION

This routine performs common I/O completion tasks. This service must be invoked by each class
driver specific status and sense action routines at the end of status and sense data processing to
perform common tasks. It can also be invoked as the default status and sense action routine if
no driver specific status and sense action is needed. The common tasks include:

- Log the I/O.
- Update `bp` with the I/O completion status.
- Cache I/O completion status and sense in the `lun` structure upon error.
- For `cmdx` and `SIOC_IO` I/Os, copy status and sense information into the `cdbinfo` structure.

RETURN VALUE

- `-1` If the I/O is to be returned to the upper layers.
- `>=0` If the I/O is to be retried.

CONSTRAINTS

- No locks are held when calling this function.
- This function does not sleep.
- This function must be invoked at the end of each class driver specific status and sense action
  routine.

EXAMPLE

```c
escsi_sense_action_t xxx_sense_list[] = {
    ...
    /* Case where escsi_action() is the default sense action routine */
    { S_GOOD, S_DEFERRED_ERROR, S_RECOVERED_ERROR,
      ESCSI_SA_ANY, ESCSI_SA_ANY, escsi_action,
      ESCSI_SA_RETRY, EAGAIN, EBI_RETRY, 0},
    ...
};
```
/ Case where escsi_action is invoked by the driver specific *
/ sense action routine */
{ S_GOOD, ESCSI_SA_ANY, S_NOT_READY, ESCSI_SA_ANY,
  ESCSI_SA_ANY, xxx_sense_action, ESCSI_SA_RETRY, EBUSY,
  EEI_NONE, 2000 },

/* Class driver specific sense action routine */
int
xxx_sense_action(struct buf *bp, intptr_t flags, intptr_t errno,
intptr_t eei, intptr_t msecs)
{
  /* Handle NOT READY check condition in a driver specific manner*/
  .
  .
  /* Invoke escsi_action() before returning */
  return escsi_action(bp, flags, errno, eei, msecs);
}

SEE ALSO
escsi_process_sense(9F)
escsi_addr_t(9S)

NAME

escsi_addr_t - Generic handle that can represent a controller, target path, or a lunpath object.

SYNOPSIS

<io/escsi_services.h>

DESCRIPTION

The escsi_addr_t structure contains information that can represent a controller, target path, or a lunpath object. An interface driver uses this structure to pass information to SCSI Services. After registering a SCSI controller with the SCSI Services using escsi_ctlr_reg, interface drivers are given the SCSI controller instance number in escsi_ifctlr_reg_t->addr.c. After registering a SCSI target path with the SCSI Services using escsi_tgt_reg, interface drivers are given the following SCSI target path nexus information in escsi_iftgt_reg_t:

• The SCSI controller number of the nexus in escsi_iftgt_reg_t->addr.c.
• The SCSI target path number of the nexus in escsi_iftgt_reg_t->addr.t.

Some SCSI Services take escsi_addr_t to identify a controller, target path, or LUN path. When invoking these SCSI Services, interface drivers must populate the fields corresponding to the appropriate object:

• escsi_addr_t->c for a SCSI controller
• escsi_addr_t->c and escsi_addr_t->t for a target path
• escsi_addr_t->c, escsi_addr_t->t, and escsi_addr_t->l for a LUN path

STRUCTURE MEMBERS

typedef struct {
    uint64_t l;
    uint64_t t;
    uint32_t c;
} escsi_addr_t;

l LUN instance number, which is the 64-bit LUN address.
t SCSI target path instance number.
c SCSI controller instance number.

RETURN VALUE

None

EXAMPLE

None

CONSTRAINTS

None

SEE ALSO

escsi_ctlr_reg(9F), escsi_get_addr(9F), escsi_if_aen(9F), escsi_tgt_reg(9F)
escsi_addr_to_node(9F)

NAME
escsi_addr_to_node - Service to convert SCSI address of a SCSI component to I/O node token.

SYNOPSIS
int escsi_addr_to_node(
    escsi_addr_t *addr,
    escsi_type_t type,
    void **node
);  

PARAMETERS
addr  The escsi_addr_t of a SCSI controller, SCSI target path, or SCSI lunpath.
type Either ESCSI_CTLR, ESCSI_TP, or ESCSI_LPT as defined in escsi_type_t.
node  Pointer to the I/O node.

DESCRIPTION
This is a SCSI Kernel Programming Interface (KPI) to get the I/O node token given SCSI address
of a SCSI object.

RETURN VALUE
EINVAL    No such SCSI object, or object type is invalid.
ESUCCESS  Object node is available in node.

CONSTRAINTS
No spinlock can be held.
The addr field must contain a non-NULL value.
The type field must be a valid type.

EXAMPLE
/* lun path to controller 1, target path 0 and LUN id 0x4001000000000000 */
escsi_addr_t  lpt_addr;

lpt_addr.c = 1;
lpt_addr.t = 0;
lpt->addr.l := 0x4001000000000000;

/* Get the lun path's node using the escsi address */
rval = escsi_addr_to_node(&addr, ESCSI_LPT, &node);
if( rval != ESUCCESS ) {
    return EINVAL;
}

SEE ALSO
escsi_type_t(9S)
NAME
escsi_aen_t - The escsi_if_aen structure for bottom-up unsolicited operations from the interface driver into the SCSI Services.

SYNOPSIS
#include <io/escsi_common.h>

PARAMETERS
typedef struct {
    uint64_t                  version;
    escsi_addr_t              addr;
    escsi_async_evt_t         evt;
    intptr_t                  evt_data;
    intptr_t                  rsvd[5];
} escsi_aen_t;

DESCRIPTION
The escsi_aen_t structure is passed as a parameter to the escsi_if_aen function.

STRUCTURE MEMBERS
version  Specifies the API version.
addr     Specifies the address of the component handling the asynchronous event.
evt      Specifies the asynchronous event.
evt_data Specifies one of the following event data:
    ESCSI_IOBJ_CPU_REG  Pointer to escsi_iobj_cpu_t.
    ESCSI_IF_OBJ_REG    Pointer to escsi_if_iobj_t.
    ESCSI_PORT_ID_CHANGE Pointer to a uint64_t port identifier.


CONSTRAINTS
None

SEE ALSO
escsi_addr_t, (9S), escsi_async_evt_t(9S), escsi_if_aen(9F)
ESCSI_APPEND_LIST(9G)

NAME

ESCSI_APPEND_LIST - Macro to append one list after another list.

SYNOPSIS

ESCSI_APPEND_LIST (escsi_list_t *__hdr1,
                escsi_list_t *__hdr2)

PARAMETERS

__hdr1  Pointer to a list header.
__hdr2  Pointer to a list header.

DESCRIPTION

The ESCSI_APPEND_LIST macro appends the list pointed to by __hdr2 to the tail of the list pointed to by __hdr1. The new list is pointed to by the __hdr1 header. The __hdr2 argument must not be an empty list during the call.

RETURN VALUE

None

CONSTRAINTS

None

EXAMPLE

escsi_list_t list1_hdr;
escsi_list_t list2_hdr;
ESCSI_APPEND_LIST(&list1_hdr, &list2_hdr);

SEE ALSO

ESCSI_PREPEND_LIST(9G)
NAME

escsi_apsw_reg - Register the Active/Passive device plug-in functions (APSW) with SCSI Services.

SYNOPSIS

#include <sys/escsi_services.h>

escsi_apsw_hdl_t escsi_apsw_reg(
    escsi_apsw_t *apsw
);

PARAMETERS

apsw    Pointer to a escsi_apsw_t structure.

DESCRIPTION

The escsi_apsw_reg function is called by Active/Passive device plug-in kernel modules to register their functions (APSW) with SCSI Services. The Active/Passive plug-in module provides plug-in functions respectively to check whether a lunpath is active or standby (driver_check_lpt), to activate a standby lunpath (driver_activate_lpt), and, if applicable, to examine device-specific sense key data sent by the Active/Passive device when the state of the lunpath changes. The plug-in module also specifies during the registration the set of Active/Passive array controllers to which the plug-in functions apply, by specifying their vendor identifiers (vid) and product identifiers (pid) as returned in inquiry data. The plug-in module must perform the registration of the plug-ins functions in its module initialization routine.

To provide the maximum flexibility for the registration of plug-in functions for devices, the registration services enable the functions to specify up to ESCSI_MAX_APSW vid/pid entries. A vid/pid entry can be specific or wild card to cover a series of device families. In a specific vid/pid entry, the vid and pid are completely specified as they might be returned in inquiry data. In a wild card vid/pid entry, the vid and pid are completely specified as they might be returned in inquiry data. For example:

vid="HP      
pid="ST39103FC       

In a wild card vid/pid entry, the vid is completely specified, but only the beginning of the pid is specified and terminated with the asterisk character (*) to mark it as a wild card entry. For example, the following wild card entry covers all active/passive array controllers from HP with a product identifier starting with ST:

vid="HP      
pid="ST*" 

The vid/pid entry specifications have the following rules:

• You must specify at least one vid/pid entry.
• The vid and pid of a specific entry must match what the device returns in the inquiry data, including the characters' case and the padding with space characters to meet the standard size of 8 and 16 characters, respectively, for the vid and pid.
• The pid of a wild card entry must cannot start with an asterisk character (*). The following example shows invalid wild card pids:

  "**
  "*ST"
  "*S*"

  The following example shows valid wild card pids:

  "S*
  "ST*"
  "ST3*"
The escsi_apsw_t structure is defined as follows:

```c
#define ESCSI_MAX_APSW 5

typedef struct {
    escsi_devid_t devid[ESCSI_MAX_APSW];
    escsi_check_lpt_t check_lpt;
    escsi_activate_lpt_t activate_lpt;
    escsi_inspect_lpt_err_t inspect_lpt_err;
} escsi_apsw_t;
```

devid

An array of type escsi_devid_t. Each element of this array specifies the vendor identification (vid), product identification (pid), and peripheral device type (pdt) for which the plug-ins are to be associated. The present implementation of escsi_apsw_reg requires specification of only the vid and pid fields in the escsi_devid_t structure. The pdt field can be left unspecified or null because this field is ignored. The maximum size of this array is ESCSI_MAX_APSW. This means that a set of plug-ins can be registered or associated with a maximum number of ESCSI_MAX_APSW devices in a single call to escsi_apsw_reg.

The escsi_devid_t structure is defined as follows:

```c
#define ESCSI_VID_SZ 8 /* maximum size of vendor identifier */
#define ESCSI_PID_SZ 16  /* maximum size of product identifier */
typedef struct {
    uint8_t pdt;
    char vid[ESCSI_VID_SZ];
    char pid[ESCSI_PID_SZ];
} escsi_devid_t;
```

**NOTE:** SCSI Services does not support null terminated character strings. The sizes of strings assigned to the vid and pid fields must be equal to the size of the character arrays. Shorter strings must be padded with blanks (ASCII value 0x20).

check_lpt

The address of the plug-in function to check the state of the lunpath (driver_check_lpt).

activate_lpt

The address of the plug-in function to activate a standby lunpath (driver_activate_lpt).

inspect_lpt_err

The address of the plug-in function to examine device specific sense key data sent when the state of a lunpath changes (driver_inspect_lpt_err).

**NOTE:** This is an optional function. You must provide one only if the active/passive device returns a check condition with device-specific sense key data when the state of the lunpath changes.

**RETURN VALUES**

A structure of type escsi_apsw_hdl_t. This is reserved for future use.

**CONSTRAINTS**

You must initialize the apsw structure to zero before each call to escsi_apsw_reg.
SEE ALSO

escsi_cdbinfo_init(9F), escsi_cmdx_ext(9F), escsi_get_inq_serial(9F), driver_activate_lpt(9F),
driver_check_lpt(9F), driver_inspect_lpt_err(9F)
escsi_async_evt_t(9S)

NAME

escsi_async_evt_t - Enumeration for asynchronous event notification.

SYNOPSIS

#include <sys/escsi_services.h>

DESCRIPTION

The escsi_async_evt_t structure contains SCSI asynchronous events or notifications that are sent by the interface driver to the SCSI Services through the escsi_if_aen KPI.

STRUCTURE MEMBERS

typedef enum {
    ESCSI_TP_ONLINE    = 0x01,
    ESCSI_TP_OFFLINE   = 0x02,
    ESCSI_LPT_ONLINE   = 0x03,
    ESCSI_LPT_OFFLINE  = 0x04,
    ESCSI_IF_OBJ_REG   = 0x05,
    ESCSI_IOBJ_CPU_REG = 0x06,
    ESCSI_PORT_ID_CHANGE = 0x07,
} escsi_async_evt_t;

ESCSI_TP_ONLINE To notify that a target path has come online. This asynchronous event is expected from interface drivers only.

ESCSI_TP_OFFLINE To notify that a target path has gone offline. Reserved for interface drivers only.

ESCSI_LPT_ONLINE To notify that a LUN path has come online. Reserved for class drivers only.

ESCSI_LPT_OFFLINE To notify that a LUN path has gone offline. Reserved for class drivers only.

ESCSI_IF_OBJ_REG To bind an I/O object with an interface driver private handle. Reserved for interface drivers only.

ESCSI_IOBJ_CPU_REG To bind a CPU to an I/O object. Reserved for interface drivers only.

ESCSI_PORT_ID_CHANGE To notify a change of a target port identifier. Reserved for interface drivers only.

SEE ALSO

escsi_if_aen(9F)
NAME
escsi_bool_t - General purpose boolean definition

SYNOPSIS
#include <sys/escsi_services.h>

DESCRIPTION
This structure contains the general purpose enumeration of boolean values used throughout the SCSI Services.

STRUCTURE MEMBERS
typedef enum {
    ESCSI_FALSE = 0x00,
    ESCSI_TRUE  = 0x01
} escsi_bool_t;

ESCSI_FALSE  Condition is not met or is not true.
ESCSI_TRUE   Condition is met or is true.

SEE ALSO
NAME
escsi_cdbinfo_alloc - Allocate and initialize a cdbinfo structure.

SYNOPSIS
escsi_cdbinfo_t *escsi_cdbinfo_alloc(
    dev_t devt,
    int dev_type,
    escsi_lun_t *lun,
    escsi_lpt_t *lpt,
    escsi_addr_t *lpt_addr,
    escsi_sctl_io_flags_t esctl_io_flags,
    escsi_cdbinfo_flags_t cdbinfo_flags,
    uint32_t cdb_len,
    uint8_t *cdb,
    uint32_t data_len,
    void *data,
    int max_msecs,
    int16_t max_retries,
    escsi_status_action_t *sa,
    int sa_size,
    escsi_cbfunc_t cb_func,
    void *cb_arg,
    intptr_t caller_ws,
    int kmalloc_flags
);

PARAMETERS

devt
The dev_t of the LUN passed by interface drivers. The caller must also set the ESCTL_LUN_BY_DEVT flag in the cdbinfo_flags parameter to indicate that the LUN information is being passed in via devt instead of a pointer.

dev_type
The device type (D_BLK or D_CHR).
lun
Pointer to the lun structure passed by class drivers.
lpt
Pointer to the lunpath structure passed by class drivers.
lpt_addr
Pointer to escsi_addr_t structure passed by interface drivers. The caller must also set the ESCTL_LPT_BY_HANDLE flag in the cdbinfo_flags parameter to indicate that the lunpath information is being passed in via escsi_addr_t instead of the lpt pointer.
esctl_io_flags
I/O type (read vs write, LUN vs lunpath)
cdbinfo_flags
I/O execution (wait vs no wait, regular vs hi-priority).
cdb_len
Specifies the length of the cdb array.
cdb
Pointer to the SCSI cdb structure.
data_len
Specifies the length of data buffer allocated by the caller.
data
Specifies the data buffer.
max_msecs
Specifies the timeout interval specified by the caller.

The caller can specify that the default timeout value be used by passing in ESCSI_IO_DFLT_TIMEOUT (-1). The default value is taken from lpt->max_msecs. The caller can also specify that the I/O must not be
timed by the interface driver by passing in ESCSI_IO_INFINITE_TIME (0).

### max_retries
Specifies the number of retries.
The caller can specify that the default number of retries be used by passing in ESCSI_IO_DFLT_RETRIES (-1). The default value is 10.

### sa
Specifies the status action list (optional). If not supplied, the status action list is taken from the lun->ddsw.

### sa_size
Specifies the status action list size.

### cb_func
Specifies the callback function (if I/O is no wait).

### cb_arg
Specifies the callback function argument.

### caller_ws
Pointer to caller’s private work space.

### kmalloc_flags
M_WAITOK or M_NOWAIT.

### DESCRIPTION
This function allocates and initializes a cdbinfo structure. The cdbinfo structure contains the details of the I/O that can be issued via the escsi_cmdx_ext interface. This interface allows the various modules in the mass storage stack (SCSI Services, class drivers, interface drivers) to issue CDBs directly to a device. This service takes the arguments necessary to issue an I/O via the escsi_cmdx_ext interface, allocates a cdbinfo structure and initializes it with given arguments. The service also performs a sanity check on the arguments.

### RETURN VALUE
On successful completion, returns a pointer to a cdbinfo structure. Otherwise, returns NULL, if memory allocation fails or the caller passed in invalid arguments.

### CONSTRAINTS
No spinlocks can be held across this call, if M_WAITOK flag is set. If M_NOWAIT is specified, the caller can hold a spinlock.

When calling the service with the ESCTL_LUN_BY_DEVT cdbinfo flag set, this service acquires an eSCSI global lock whose lock order is ESCSI_GLOBAL_LOCK_ORDER. Therefore, the caller is not expected to hold locks of order greater than or equal to ESCSI_GLOBAL_LOCK_ORDER when passing the LUN information by devt (ESCTL_LUN_BY_DEVT)

### SEE ALSO
escsi_addr_t(9S), escsi_cdbfunc_t(9S), escsi_cdbinfo_flags_t(9S), escsi_lpt_t(9S), escsi_lun_t(9S), escsi_sctl_io_flags_t(9S), escsi_status_action_t(9S)
NAME
escsi_cbfn_t - Generic callback parameter prototype.

SYNOPSIS
#include <io/escsi_common.h>

DESCRIPTION
This structure provides a pointer to a callback function and a pointer to the arguments to the
callback function.

STRUCTURE MEMBERS
typedef struct {
    escsi_cbfunc_t         cb_func;
    void                        *cb_arg;
} escsi_cbfn_t;

cb_func    Specifies the callback function.
cb_arg     Specifies the callback argument.

RETURN VALUE
None

EXAMPLE
None

CONSTRAINTS
None

SEE ALSO
escsi_cdbfunc_t(9S), escsi_cdbinfo_t(9S), escsi_ifspoc_ws_t(9S)
escsi_cbfunc_t(9S)

NAME
escsi_cbfunc_t - Callback function prototype.

SYNOPSIS
#include <io/escsi_common.h>

DESCRIPTION
The callback function prototype. Callback functions are commonly used in asynchronous KPI (for example, escsi_tgt_reg).

STRUCTURE MEMBERS
typedef void (*escsi_cbfunc_t)(void *, int);
The input parameters are the callback argument and errno value.

RETURN VALUE
None

EXAMPLE
None

CONSTRAINTS
None

SEE ALSO
escsi_cbfn_t(9S), escsi_iftgt_reg_t(9S)
NAME
escsi_cdbinfo_flags_t - Enumerations for the Command Data Block (CDB) structure flags

SYNOPSIS
#include <io/escsi_common.h>

DESCRIPTION
Data structures and type definitions related to cdbinfo structure that specify how the I/O is to be handled by the SCSI Services. These are bit flags. The cdbinfo structure is used to issue requests via the escsi_cmdx and escsi_cmdx_ext service.

STRUCTURE MEMBERS
typedef enum {
    ESCTL_IO_NOWAIT = 0x1,
    ESCTL_SEND_ALWAYS = 0x2,
    ESCTL_CBARG_CDBINFO = 0x4,
    ESCTL_LPT_BY_HANDLE = 0x8,
    ESCTL_LUN_BY_DEVT = 0x10
} escsi_cdbinfo_flags_t;

ESCTL_IO_NOWAIT Asynchronous I/O. The caller does not wait for completion of the I/O.
ESCTL_SEND_ALWAYS I/O must be treated as high priority I/O and issued bypassing flow controls. The SCBF_HIPRI flag is set in the escb structure associated with the I/O.
ESCTL_CBARG_CDBINFO Caller wants the callback argument to be cdbinfo pointer.
ESCTL_LPT_BY_HANDLE Indicates the caller is passing the lunpath information in c-t-l format (escsi_addr_t). Absence of this flag indicates the caller is passing the lunpath by pointer, which is the default mechanism. This flag must only be set when issuing path-specific I/O.
ESCTL_LUN_BY_DEVT Indicates caller is passing lun information by devt. The default mechanism is to pass information via lun pointer.

SEE ALSO
escsi_cmdx(9F), escsi_cmdx_ext(9F)
escsi_cdbinfo_free(9F)

NAME
escsi_cdbinfo_free - Free a cdbinfo structure.

SYNOPSIS

void escsi_cdbinfo_free(
    escsi_cdbinfo_t *cdbinfo
);

PARAMETERS
cdbinfo Pointer to cdbinfo structure

DESCRIPTION
This function is used to deallocate a cdbinfo structure.

RETURN VALUE
None

CONSTRAINTS
No locks must be held while calling this function.

SEE ALSO
escsi_cdbinfo_t(9S)
NAME

escsi_cdbinfo_init - Initialize a Command Descriptor Block (CDB) structure.

SYNOPSIS

void escsi_cdbinfo_init(
    escsi_cdbinfo_t *cdbinfo,
    dev_t devt,
    int dev_type,
    escsi_lun_t *lun,
    escsi_lpt_t *lpt,
    escsi_addr_t *lpt_addr,
    esctl_io_flags_t esctl_io_flags,
    escsi_cdbinfo_flags_t cdbinfo_flags,
    uint32_t cdb_len,
    uint8_t *cdb,
    uint32_t data_len,
    void *data,
    int max_msecs,
    int16_t max_retries,
    escsi_status_action_t *sa,
    int sa_size,
    escsi_cbfunc_t cb_func,
    void *cb_arg,
    intptr_t caller_ws,
    int kmalloc_flags
);

PARAMETERS

cdbinfo Pointer to the cdbinfo structure to be initialized.
devt The dev_t of the LUN passed by interface drivers.
dev_type The type of devt (D_BLK or D_CHR), set by caller.
lun Pointer to the lun structure passed by class drivers.
lpt Pointer to the lunpath structure passed by class drivers.
lpt_addr Pointer to escsi_addr_t structure passed by interface drivers.
esctl_io_flags Specifies the nature of the I/O. The value passed in must be one of the following:
ESCTL_I/O_LPT Indicates that this is a path-specific I/O.
ESCTL_READ Indicates that this is a read-type I/O.

cdbinfo_flags I/O execution (wait vs no wait, regular vs hi-priority).
cdb_len Specifies the length of the cdb array.
cdb Pointer to the CDB of the command being issued.
data_len Specifies size (in bytes) of the data transfer, if required for this specific CDB. The transfer size cannot exceed 512. The argument must be zero if no data transfer is required.
data Specifies the data buffer.
max_msecs Specifies the I/O timeout value. If you do not explicitly specify a timeout value, you can specify one of the following values:
ESCSI_I/O_DFLT_TIMEOUT The default timeout value is sufficient.

ESCSI_I/O_INFINITE_TIME Do not time the I/O.

max_retries Specifies the maximum number of I/O retries. If you do not explicitly specify a retry value, you can specify the following:

ESCSI_I/O_DFLT_RETRIES The default number of retries is required.

sa Specifies the status action list (optional).

sa_size Specifies the status action list size.

cb_func Specifies the callback function to be called when I/O completes. Before using this field, you must back up the existing contents of the cbfn field of cdbinfo data in the cbfn_ws field of cdbinfo data.

cb_arg Specifies the callback function argument.

caller_ws Pointer to caller’s private work space.

kmalloc_flags M_WAITOK or M_NOWAIT.

DESCRIPTION

The escsi_cdbinfo_init function allocates and initializes a cdbinfo structure. The cdbinfo structure contains the details of the I/O that can be issued via the escsi_cmdx_ext interface. This interface allows the various modules in the mass storage stack (SCSI Services, class drivers, and interface drivers) to issue CDBs directly to a device. This service takes the arguments necessary to issue an I/O via the escsi_cmdx_ext interface, allocates a cdbinfo structure and initializes it with given arguments. The service also validates the arguments.

The escsi_cdbinfo_init function can also reinitialize a cdbinfo structure that was previously allocated by calling escsi_cdbinfo_alloc. It allows the caller to reuse the same cdbinfo structure to issue another CDB.

RETURN VALUE

None

CONSTRAINTS

No locks must be held while calling this function.

As part of initialization or reinitialization, do not initialize the cdbinfo structure to zero.

EXAMPLE

A typical call to escsi_cdbinfo_init is as follows. The data that can be modified are identified:

cdbinfo->cbfn_ws = cdbinfo->cbfn;
escsi_cdbinfo_init(
    cdbinfo,
    cdbinfo->devt,
    cdbinfo->lun,
    cdbinfo->lpt,
    cdbinfo->lpt_addr,
    esctl_io_flags,
    cdbinfo->cdbinfo_flags,
    cdb_len,
    cdb,
    data_len,
    cdbinfo->esioc_io->data,
    max_msecs,
    max_retries,
    cdbinfo->sa,
cdbinfo->sa_size,
cb_func,
cdbinfo,
cdbinfo->caller_ws
);

Parameter can be modified.

SEE ALSO
escsi_addr_t(9S), escsi_cdbfunc_t(9S), escsi_cdbinfo_alloc(9F), escsi_cdbinfo_flags_t(9S),
escsi_cmdx_ext(9F), escsi_lpt_t(9S), escsi_lun_t(9S), escsi_sctl_io_flags_t(9S), escsi_status_action_t(9S)
NAME
escsi_cdbinfo_t -- The ESCSI CDB information structure.

SYNOPSIS
#include <io/escsi_services.h>

DESCRIPTION
The escsi_cdbinfo_t structure encapsulates the details of the I/O being issued via the
escsi_cmdx_ext interface. The callers of the escsi_cmdx_ext interface are expected to be
native class drivers and all interface drivers.

STRUCTURE MEMBERS

typedef struct {
    int                     version;
    esctl_io_t              *esioc_io
    escsi_cdbinfo_flags_t   flags;
    int16_t                 retries;
    escsi_status_action_t   *sa;
    int                     sa_size;
    dev_t                   dev_t;
    int                     dev_type;
    escsi_lun_t             *lun;
    escsi_lpt_t             *lpt;
    escsi_addr_t            lpt_addr;
    escsi_cbfn_t            cbfn;
    struct buf              *bp;
    short                   errno;
    uint16_t                eei;
    intptr_t                caller_ws;
    escsi_cbfn_t            cbfn_ws;
    intptr_t                rsvd[5];
} escsi_cdbinfo_t;

version       The structure version.
esioc_io      Pointer to the structure containing cdb, max_msecs, data address, and I/O flags.
               On return, contains cdb status, sense status, and sense data.
flags         Specifies the cdbinfo flags.
retries       Specifies the retry to be used. A value of 0 implies the I/O must not be retried. A
               value of -1 implies the I/O must be retried the default number of times.
sa            Pointer to the status action list to be used for processing I/O completion. A NULL
               value implies the class driver status list, based on the ddsw bound to the lun.
sa_size       Specifies the size of the status action list.
dev_t         Specifies the devt of the lun. The caller specifies the lun via the devt. This field
               is set by the interface driver, which does not have access to the lun pointer.
dev_type      Specifies the type of device (D_BLK or DCHR).
lun           Pointer to the lun. The caller can specify a pointer to the lun if it has the lun
               pointer, instead of devt. This field is expected to be used by the SCSI Services
               and class drivers that deal with luns.
**lpt**  
Pointer to the lunpath. This field is set when the caller issues a path specific I/O. Because the lunpath pointer is only available to SCSI Services and class drivers, only these modules set this field.

**lpt_addr**  
Specifies the lunpath handle to be set when the interface driver needs to issue a path specific I/O.

**cbfn**  
Pointer to the callback function and its argument to be invoked, when an asynchronous I/O completes.

**bp**  
This field is reserved for use by the cdbinfo services.

**errno**  
On return, contains an error number.

**eei**  
Extended Error Information

**caller_ws**  
Specifies the private caller’s workspace. Allows the caller to attach context information that can be used in the callback function.

**cbfn_ws**  
Specifies a private field that can be used to backup `cdbinfo->cbfn` if required.

**rsvd**  
Reserved.

**SEE ALSO**

`buf(9S), escsi_addr_t(9S), escsi_cbfn_t(9S), escsi_cdbinfo_flags_t(9S), escsi_lpt_t(9S), escsi_lun_t(9S), escsi_status_action_t(9S)`
NAME
escsi_cmdx - Service for easy migration from usage of the legacy scsi_cmdx interface.

SYNOPSIS
int escsi_cmdx(
    dev_t dev,
    int dev_type,
    uint32_t flags,
    int cdb_len,
    uint8_t *cdb,
    int nbytes,
    void *addr,
    int32_t msecs,
    int32_t retries,
    int *pErr,
    escsi_status_action_t *sa,
    int n,
    escsi_lpt_t *lpt,
    escsi_cdbinfo_flags_t cdbinfo_flags
);

PARAMETERS
dev The devt of the LUN.
dev_type Type of device (D_BLK or D_CHR).
flags Specifies if this is a read (ESCTL_READ or ESCTL_IO_LPT).
cdb_len Length of the CDB passed in.
cdb Pointer to the cdbinfo structure.
nbytes Length of the data buffer.
addr Pointer to the data buffer address.
msecs I/O timeout value.
retries Number of retries.
pErr Outbound I/O status.
sa Pointer to the status action list.
n Length of the status action list.
lpt Pointer to the lunpath structure, if path-specific I/O is to be issued.
cdbinfo_flags Flags to specify SEND ALWAYS, SPECIAL, and cdbinfo attributes.

DESCRIPTION
The escsi_cmdx service allows modules in the mass storage stack to issue CDBs directly to a device. The escsi_cmdx routine allows the following I/O request types:

Asynchronous I/O request The escsi_cmdx routine returns to the caller before the I/O request completes. The caller is notified of the I/O completion via the callback provided in the cdbinfo structure.

Synchronous I/O request The escsi_cmdx service waits for the request to complete.

Both the above requests can be issued to a lun or a lunpath.
RETURN VALUE

-1    I/O did not succeed.

>=0    I/O succeeded, and indicates the number of bytes transferred.

CONSTRAINTS

The caller is expected to execute a `ESCSI_HOLD` on the lunpath if the caller needs a safeguard against a lunpath deletion.

Unless the `ESCTL_IO_NOWAIT` flag is set, the caller is expected to call this routine in a blocking context to allow waiting for the I/O to complete.

Unless `ESCTL_IO_NOWAIT` flag is set, do not hold any spinlocks across this call.

This service is for internal mass storage stack modules (class drivers, interface drivers, and SCSI Services), and only expects kernel addresses for data and sense buffers.

SEE ALSO

`escsi_cdbinfo_flags_t(9S), escsi_lpt_t(9S), escsi_status_action_t(9S)`
NAME

escsi_cmdx_ext - Issue a Command Data Block (CDB) to a device.

SYNOPSIS

void escsi_cmdx_ext(
    escsi_cdbinfo_t *cdbinfo
);

PARAMETERS

cdbinfo Pointer to a cdbinfo structure.

DESCRIPTION

The escsi_cmdx_ext service allows modules in the mass storage stack to issue CDBs directly
to a device. Before making a call to escsi_cmdx_ext, call escsi_cdbinfo_init to initialize
the CDB and other data in the cdbinfo structure.

The escsi_cmdx_ext allows the following I/O request types:

Asynchronous I/O request The escsi_cmdx_ext routine returns to the caller before
the I/O request completes. The caller is notified of the I/O
completion via the callback provided in the cdbinfo
structure.

Synchronous I/O request The escsi_cmdx_ext routine waits for the request to
complete.

Both the above requests can be issued to a lun or a lunpath.

RETURN VALUE

None

CONSTRAINTS

The caller is expected to execute a ESCSI_HOLD on the lunpath if you need to safeguard against
lunpath deletion.

Unless the ESCTL_IO_NOWAIT flag is set, the caller is expected to call this routine in a blocking
context to allow waiting for the I/O to complete.

Unless ESCTL_IO_NOWAIT flag is set, no spinlocks must be held across this call.

This service is for internal mass storage stack modules (class drivers, interface drivers, and SCSI
Services), and only expects kernel addresses for data and sense buffers.

SEE ALSO

escsi_cdbinfo_t(9S)
escsi_ctlr_node_cb(9F)

NAME
escsi_ctlr_node_cb - Controller I/O node callback function

SYNOPSIS
int escsi_ctlr_node_cb(
    void *handle,
    io_events_t event,
    uintptr_t event_info
);

PARAMETERS
handle Specifies the controller I/O node token.
event Specifies the event passed in by I/O infrastructure.
event_info Specifies the event specific data passed in by I/O infrastructure.

DESCRIPTION
This function is called by interface driver to process controller I/O node callback events related to controller scan. As part of driver initialization, interface driver would have registered its own I/O node callback function by calling wsio_claim_node. This driver I/O node callback function is called by I/O infrastructure to perform controller scan. Interface drivers must call the escsi_ctlr_node_cb SCSI Service whenever its driver I/O node callback function is invoked by I/O infrastructure with the CB_SCAN_ALL event (start of controller scan operation).

RETURN VALUE
GIO_SUCCESS Operation completed successfully.
GIO_E_FUNC_NOT_SUPPORTED Operation is not supported.

CONSTRAINTS
No spinlocks must be held while calling this function.

EXAMPLE
In this example, an interface driver controller I/O node callback routine invokes escsi_ctlr_node_cb to handle a CB_SCAN_ALL event:

```c
xxx_drv_ctlr_node_cb(void *handle,
    void *arg,
    io_events_t event,
    uintptr_t event_info)
{
    switch(event) {
    case CB_SCAN_ALL:
        /* Forward the event to the SCSI Services */
        escsi_ctlr_node_cb(handle, event, event_info);
        break;
    case CB_MK_DEV:
        /* Interface driver must handle this op-code to create ctlr dsf */
        .
        .
    case CB_GET_DEVS:
        /* Interface driver must handle this op-code to return to * IOI the devs associated with its ctlr IO node */
        .
```
SEE ALSO

escsi_tgt_node_cb(9F), wsio_node_claim(9F)
**NAME**
escsi_ctlr_reg - SCSI controller registration interface

**SYNOPSIS**

```c
int escsi_ctlr_reg(
    escsi_ifctlr_reg_t *ifctlr_reg
);
```

**PARAMETERS**

*ifctlr_reg* Pointer to controller registration structure allocated by interface driver.

**DESCRIPTION**

This function is called by an interface driver to register its controller with SCSI stack. This function can be called during interface driver initialization from the driver attach or init function. Before calling this, interface driver is expected to allocate and initialize the `escsi_ifctlr_reg_t`, `escsi_ifctlr_attr_t`, `escsi_ifctlr_stat_t`, and `escsi_ifsw_t` structures. All fields marked as “IN” in the data structure definitions must be initialized by interface driver before calling this function.

The interface driver must pass an `escsi_ifsw_t` structure with at least the `if_lpt_open`, `if_lpt_close`, `if_start`, and `if_task_mgmt` entry points defined. Because interface drivers must support the `ABORT_TASK_SET` task management option, they cannot disable it by setting this flag in the `if_tm_dsbl` field of the `escsi_ifsw_t` structure. If the interface driver uses a single I/O object per controller, it must specify `num_iobj` as 1 in `escsi_ifctlr_attr_t`.

**RETURN VALUE**

- **ESUCCESS** Registration succeeded.
- **EINVAL** Registration failed or parameter passed is invalid.

Other `errno` values as returned by `escsi_reg_node`.

**CONSTRAINTS**

This call must be made in a blocking context without holding any spinlocks. It must be called by interface driver after completing the call to `wsio_claim_node` to claim the controller.

**EXAMPLE**

The following example contains a mix of C code and pseudo code.

```c
xxx_attach() {
    
    /* Steps to register the controller with SCSI Services */
Allocate the following controller structures:
- escsi_ifctlr_reg_t
- escsi_ifsw_t
- escsi_ifctlr_attr_t
- escsi_ifctlr_stat_t
Initialize these structures with appropriate controller information.
status = escsi_ctlr_reg(ifctlr_reg);
If registration successful, ifctlr_reg->addr.c contains the
controller software identifier assigned by the SCSI Services.
This software identifier shall be used by the interface driver
when building a `escsi_addr_t` to invoke a SCSI service.
```

else
deallocate controller registration related structures

}


SEE ALSO

prop_create(9F), wsio_claim_node(9F)
NAME
escsi_ctlr_unreg - SCSI controller unregistration interface

SYNOPSIS
int escsi_ctlr_unreg(
    escsi_ifctlr_reg_t *ifctlr_reg
);

PARAMETERS
ifctlr_reg Pointer to controller registration structure allocated by interface driver.

DESCRIPTION
This function is called by an interface driver to unregister its controller which was previously registered with SCSI stack. It is called by the interface driver during an Online Deletion (OLD) or Dynamically Loaded Kernel Module (DLKM) unload operation. Before making this call, the interface driver must unregister all the target paths registered under this controller from SCSI Services. On return from this call, the interface driver is expected to deallocate the escsi_ifctlr_reg_t structure and other related structures linked to it.

RETURN VALUE
ESUCCESS Unregistration succeeded.
EINVAL Invalid parameter passed or unregistration failed.

CONSTRAINTS
The call must be made in a blocking context without holding any spinlocks.

EXAMPLE
The following example contains a mix of C code and pseudo code.

```c
xxx_uninit() { 
    
    For every registered tgtpath {
        Unregister tgtpath by calling escsi_tgt_unreg()
        If unregistration successful, deallocate target related structures
    }
    /* Steps involved in unregistering the SCSI ctlr component */
    Call escsi_ctlr_unreg(escsi_ifctlr_reg_t *ifctlr_reg)

    If unregistration successful {
        Deallocate the following structures:
        - escsi_ifctlr_reg_t
        - escsi_ifsw_t
        - escsi_ifctlr_attr_t
        - escsi_ifctlr_stat_t
    } else
        VASSERT
    
}
```
SEE ALSO

prop_destroy(9F)
**escsi_dd_reg(9F)**

**NAME**

escsi_dd_reg - Register a class driver with the SCSI Services

**SYNOPSIS**

```c
int escsi_dd_reg(
    escsi_ddsw_t *ddsw
);
```

**PARAMETERS**

*ddsw*  
Pointer to `escsi_ddsw_t` structure.

**DESCRIPTION**

This routine registers a class driver `escsi_ddsw_t` structure with SCSI Services. Class drivers must invoke it in the `driver_install` entry point to further interface with SCSI Services.

**RETURN VALUES**

ESUCCESS  
The registration was successful.

EINVAL  
The `ddsw->wsioDrvInfo` value is NULL.

**CONSTRAINTS**

This routine must be invoked in the `driver_install` entry point.

**EXAMPLE**

```c
void mydriver_install(void)
{
    /* Register ddsw with scsi services */
    if(escsi_dd_reg(&mydriver_ddsw) != ESUCCESS) {
        ESCSI_ASSERT(ESCSI_FALSE);
    }

    /* Register class driver attributes */
    .
    .
    /* Register with wsio */
    wsio_install_driver(&eschgr_wsio_info);
}
```

**SEE ALSO**

`escsi_ddsw_t(9S), escsi_vpid_reg(9F)`
### NAME
escsi_ddsw_t - Structure for registering a class driver with SCSI Services

### SYNOPSIS

#### STRUCTURE MEMBERS

```c
typedef struct escsi_ddsw {
    escsi_list_t     list;
    escsi_list_t     vidpid_list;
    uint64_t         version;
    uint32_t         dd_lun_size;
    uint32_t         dd_lpt_size;
    void             (*dd_lpt_init)__((escsi_lpt_t *lpt));
    void             (*dd_lpt_uninit)__((escsi_lpt_t *lpt));
    void             (*dd_lun_init)__((escsi_lun_t *lun, escsi_lpt_t *lpt));
    void             (*dd_aen)__((void *hdl, escsi_async_evt_t event,
                                intptr_t event_arg));
    int              (*dd_strategy)__((escsi_lun_t *lun, struct buf *bp));
    struct buf       *(*dd_start)__((escsi_lun_t *lun));
    void             (*dd_io_init)__((escsi_lpt_t *lpt,
                                      struct buf *bp,
                                      escsi_scb_t *scb));
    int              (*dd_rsvd1)__((void));
    int              (*dd_rsvd2)__((void));
    void             (*dd_done)__((struct buf *bp));
    escsi_bool_t     (*dd_pass_thru_okay)__((escsi_cdbinfo_t *cdbinfo));
    void             (*dd_pass_thru_done)__((struct buf *bp));
    int              (*dd_ioctl)__((dev_t dev, escsi_lun_t *lun,
                                   escsi_lpt_t *lpt, int cmd,
                                   caddr_t data, int flags));
    escsi_bool_t     (*dd_ioctl_okay)__((dev_t dev, escsi_lpt_t *lpt,
                                      int cmd, caddr_t data,
                                      int flags));
    escsi_status_action_t *dd_status_list;
    int              dd_status_cnt;
    wsio_drv_info_t  *wsio_drv_info;
    wsio_drv_info_t  *leg_wsio_drv_info;
    intptr_t         rsvd[8];
} escsi_ddsw_t;
```

- **list**
  - Private to SCSI Services.
- **vidpid_list**
  - A list of VID, PID, and PDT information that is used to bind luns to third-party drivers. It is populated by a call to `escsi_vidpid_reg`.
- **version**
  - Version of the structure.
- **dd_lun_size**
  - Specifies the number of bytes the device driver wants SCSI Services to allocate on its behalf for its private LUN structure.
- **dd_lpt_size**
  - Specifies the number of bytes the device driver wants SCSI Services to allocate on its behalf for its private lunpath structure.
- **dd_lpt_init**
  - Invoked by SCSI Services to allow class driver specific configuration of a discovered lunpath.
- **dd_lpt_uninit**
  - Invoked by SCSI Services to allow class driver specific unconfiguration of a lunpath in one of the following contexts: lunpath removal, target path removal, or controller removal with the `rmsf` command or online deletion (OLD).
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dd_lun_init</code></td>
<td>Invoked by SCSI Services to allow class driver specific configuration of a discovered LUN. This entry point is invoked before <code>dd_lpt_init</code>.</td>
</tr>
<tr>
<td><code>dd_aen</code></td>
<td>Invoked by SCSI Services to handle a lunpath going offline or coming back online in a class driver specific manner. The events are <code>ESCSI_LPT_OFFLINE</code> and <code>ESCSI_LPT_ONLINE</code>.</td>
</tr>
<tr>
<td><code>dd_strategy</code></td>
<td>Invoked by SCSI Services to perform device driver I/O checking (for example, block alignment) and to queue I/O in a class driver. It returns 0 for success or non-0 for failure. The class driver must fail the I/O back to the upper layers by calling <code>escsi_iiodone</code>. This entry point is mandatory for all class drivers.</td>
</tr>
<tr>
<td><code>dd_start</code></td>
<td>The class driver returns a dequeued I/O from its internal queue, previously queued in <code>dd_strategy</code>. This entry point is mandatory.</td>
</tr>
<tr>
<td><code>dd_io_init</code></td>
<td>Invoked by SCSI Services to initialize the <code>scb</code> from the information in the <code>bp</code>. In particular, setup a read or write a CDB. This entry point is mandatory.</td>
</tr>
<tr>
<td><code>dd_rsvid1</code></td>
<td>Reserved</td>
</tr>
<tr>
<td><code>dd_rsvid2</code></td>
<td>Reserved</td>
</tr>
<tr>
<td><code>dd_done</code></td>
<td>SCSI Services calls this routine to notify the class driver before a non pass-through I/O is completed using <code>biodone</code>. The class driver can examine the <code>scb</code> and record any I/O completion information. This entry point is optional.</td>
</tr>
<tr>
<td><code>dd_pass_thru_okay</code></td>
<td>SCSI Services calls this entry point to query whether a pass-through I/O can be issued on the device. The class driver must provide this function if it needs to control pass-through I/Os and <code>escsi_cmdx</code> I/Os that can be issued on the LUN. If a driver does not provide this function, pass through I/Os and <code>escsi_cmdx</code> I/Os are allowed unconditionally. This is an optional entry point.</td>
</tr>
<tr>
<td><code>dd_pass_thru_done</code></td>
<td>The SCSI Services calls this entry point to notify a class driver of a pass-through I/O completion. This enables the class driver to check the pass-through I/O completion status or any sense data. This entry point is optional.</td>
</tr>
<tr>
<td><code>dd_ioctl</code></td>
<td>SCSI Services calls this entry point to handle native class driver ioctl. This entry point is optional.</td>
</tr>
<tr>
<td><code>dd_ioctl_okay</code></td>
<td>For reasons similar to those justifying <code>dd_pass_thru_okay</code>, device drivers are given control over which ioctl are executed by the pass-through driver while they have a device open. Unlike <code>dd_pass_thru_okay</code>, only ioctl generated with the pass-through driver are subject to <code>dd_ioctl_okay</code>. Others are subject to <code>d_ioctl</code>. As with <code>dd_pass_thru_okay</code>, <code>dd_ioctl_okay</code> is optional. If omitted, it implies the class driver permits all ioctl. This entry point is optional.</td>
</tr>
<tr>
<td><code>dd_status_list</code></td>
<td>Pointer to a list of status actions.</td>
</tr>
<tr>
<td><code>dd_status_cnt</code></td>
<td>Specifies the number of status actions.</td>
</tr>
<tr>
<td><code>wsio_drv_info</code></td>
<td>Pointer to the driver <code>wsio_drv_info_t</code> structure.</td>
</tr>
<tr>
<td><code>leg_wsio_drv_info</code></td>
<td>Pointer to the legacy driver <code>wsio_drv_info_t</code> structure.</td>
</tr>
<tr>
<td><code>rsvid</code></td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
DESCRIPTION
The SCSI Services layer uses the escsi_ddsw_t SCSI device driver switch structure for interacting with class drivers. This structure contains several interface functions and attribute values specific to the class driver.

CONSTRAINTS
The device driver switch structure must be registered using escsi_dd_reg in the driver_install entry point.

EXAMPLE
See the escsi_dd_reg(9F) example.

SEE ALSO
escsi_dd_reg(9F), escsi_vidpid_reg(9F)
NAME
ESCSI_DEQ - Macro to dequeue a linked list element.

SYNOPSIS
ESCSI_DEQ(escsi_list_t *__elem)

PARAMETERS
__elem Pointer to list element to be dequeued.

DESCRIPTION
The ESCSI_DEQ macro dequeues a given list element from the linked list in which it is present.

RETURN VALUE
None

CONSTRAINTS
None

EXAMPLE
escsi_list_t elem;
ESCSI_DEQ(&elem);

SEE ALSO
NAME
escsi_deq_active_lpt - Dequeue a lunpath from the active path queue

SYNOPSIS

```c
void escsi_deq_active_lpt(
    escsi_lun_t *lun,
    escsi_lpt_t *lpt
);
```

PARAMETERS

- `lun` Pointer to the LUN structure.
- `lpt` Pointer to the lunpath element to dequeue from the active lunpath queue.

DESCRIPTION

The `escsi_deq_active_lpt` service dequeues a lunpath from the LUN active path queue.

RETURN VALUE

None

CONSTRAINTS

The caller must hold the LUN lock.

The `lpt` that is requested to be dequeued must be on the LUN’s active path queue.

EXAMPLE

```c
void mydriver_dd_aen(void *lpt_hdl, escsi_async_evt_t event, intptr_t ev_arg)
{
    switch (event) {
    case ESCSI_LPT_OFFLINE:
        escsi_lpt_offline(lpt);
        /* Dequeue path from active path queue */
        ESCSI_LUN_LOCK(lun);
        /* Dequeue offline path from lun active path queue */
        if (escsi_is_on_q(&lun->active_path_q,
            (escsi_list_t *)lpt) == ESCSI_TRUE) {
            escsi_deq_active_lpt(lun, lpt);
        }
        /* Class driver-specific lunpath offline action */
        .
        .
        /* Compute path counts for the lun */
        escsi_lun_compute_path_cnt(lun);
        ESCSI_LUN_UNLOCK(lun);
        break;
    }
    .
    .
    return;
}
```
SEE ALSO

escsi_enq_active_lpt(9F), escsi_is_on_q(9F), escsi_lpt_offline(9F), escsi_lun_compute_path_cnt(9F)
**NAME**

ESCSI_DEQ_BP - Macro to dequeue a buffer structure from anywhere in a list.

**SYNOPSIS**

```
ESCSI_DEQ_BP(__qh, __bp)
```

**PARAMETERS**

- `__qh` Pointer to the head of the queue.
- `__bp` Pointer to a `buf` structure element in the queue.

**DESCRIPTION**

The `ESCSI_DEQ_BP` macro dequeues a `buf` structure from any position in a doubly linked list.

**RETURN VALUES**

None

**CONSTRAINTS**

None

**EXAMPLE**

```c
struct buf *buf_q; /* Head of bp queue */
struct buf *bp;

/* Assume that bp element is queued in buf_q_queue.
 * Dequeue bp element.
 */
ESCSI_DEQ_BP(&buf_q, bp);
```

**SEE ALSO**

`ESCSI_DEQ_BP_FROM_HEAD(9G), ESCSI_DEQ_BP_FROM_HEAD_FAST(9G)`
ESCSI_DEQ_BP_FROM_HEAD(9G)

NAME

ESCSI_DEQ_BP_FROM_HEAD - Macro to dequeue a buf structure element from the head of a doubly linked list

SYNOPSIS

ESCSI_DEQ_BP_FROM_HEAD(__qh, __bp)

PARAMETERS

__qh Pointer to the head of the queue.
__bp Pointer to a buf structure element in the queue.

DESCRIPTION

The ESCSI_DEQ_BP_FROM_HEAD macro dequeues a buf structure from the head of a doubly linked list.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

struct buf  *buf_q; /* Head of bp queue */
struct buf  *bp;

/*
 * Dequeue the first bp element from the queue.
 */
ESCSI_DEQ_BP_FROM_HEAD(&buf_q, bp);

SEE ALSO

dd_fast(9F), ESCSI_DEQ_BP(9G), ESCSI_DEQ_BP_FROM_HEAD_FAST(9G)
ESCSI_DEQ_BP_FROM_HEAD_FAST(9G)

NAME
ESCSI_DEQ_BP_FROM_HEAD_FAST - Macro to dequeue a buf structure element from the head of a doubly linked list.

SYNOPSIS
ESCSI_DEQ_BP_FROM_HEAD_FAST(__qh, __bp)

PARAMETERS
__qh  Pointer to the head of the queue.
__bp  Pointer to a buf structure element in the queue.

DESCRIPTION
The ESCSI_DEQ_BP_FROM_HEAD_FAST macro dequeues a buf structure from the head of a doubly linked list.

RETURN VALUES
None

CONSTRAINTS
The macro assumes the list is non empty.

EXAMPLE
struct buf *buf_q;    /* Head of bp queue */
struct buf *bp;

/*
 * Dequeue the first bp element from the queue.
 */
ESCSI_DEQ_BP_FROM_HEAD_FAST(&buf_q, bp);

SEE ALSO
ESCSI_DEQ_BP(9G), ESCSI_DEQ_BP_FROM_HEAD(9G)

886  SCSI Reference Pages
ESCSI_DEQ_FROM_HEAD(9G)

NAME
ESCSI_DEQ_FROM_HEAD - Macro to dequeue a linked list element from head of list.

SYNOPSIS
ESCSI_DEQ_FROM_HEAD( escsi_list_t *__hdr,
    escsi_list_t *__elem )

PARAMETERS
__hdr        Pointer to linked list header.
__elem       Pointer to list element that is dequeued.

DESCRIPTION
The ESCSI_DEQ_FROM_HEAD macro dequeues a given list element from the linked list in which it is present. This macro takes as an argument a pointer to the linked list header and __elem, which is of type escsi_list_t *. The address of the list element dequeued is placed in __elem. If the list is empty when this macro is called, __elem contains NULL as result.

RETURN VALUE
None

CONSTRAINTS
None

EXAMPLE
escsi_list_t elem;
escsi_list_t header;
ESCSI_DEQ_FROM_HEAD(&header, &elem);

SEE ALSO
ESCSI_DEQ_FROM_HEAD_FAST(9G)

NAME

ESCSI_DEQ_FROM_HEAD_FAST - Macro to dequeue a linked list element from head of list.

SYNOPSIS

ESCSI_DEQ_FROM_HEAD_FAST( escsi_list_t *__hdr,
                        escsi_list_t *__elem )

PARAMETERS

__hdr    Pointer to linked list header.
__elem   Pointer to list element that is dequeued.

DESCRIPTION

The ESCSI_DEQ_FROM_HEAD_FAST macro dequeues a list element from the head or tail of a given linked list, when it is known that the linked list is not empty; the macro does not check whether the list is empty. This macro takes as an argument a pointer to the linked list header __hdr and __elem, which is of type escsi_list_t *. The address of the list element dequeued is placed in __elem.

RETURN VALUE

None

CONSTRAINTS

Do not use this macro when the list is empty.

EXAMPLE

escsi_list_t elem;
escsi_list_t header;
ESCSI_DEQ_FROM_HEAD_FAST(&header, &elem);

SEE ALSO
NAME
ESCSI_DEQ_FROM_TAIL - Macro to dequeue a linked list element from tail of list.

SYNOPSIS
ESCSI_DEQ_FROM_TAIL (escsi_list_t *__hdr,
                        escsi_list_t *__elem )

PARAMETERS
__hdr Pointer to linked list header.
__elem Pointer to list element that is dequeued.

DESCRIPTION
The ESCSI_DEQ_FROM_TAIL macro dequeues a given list element from the linked list in which
it is present. This macro takes as an argument a pointer to the linked list header and __elem,
which is of type escsi_list_t *. The address of the list element dequeued is placed in __elem.
If the list is empty when this macro is called, __elem contains NULL as result.

RETURN VALUE
None

CONSTRAINTS
None

EXAMPLE
escsi_list_t elem;
escsi_list_t header;
ESCSI_DEQ_FROM_TAIL(&header, &elem);

SEE ALSO
NAME

ESCSI_DEQ_FROM_TAIL_FAST - Macro to dequeue a linked list element from tail of list.

SYNOPSIS

ESCSI_DEQ_FROM_TAIL_FAST (escsi_list_t *__hdr,
                         escsi_list_t *__elem )

PARAMETERS

__hdr     Pointer to linked list header.
__elem    Pointer to list element that is dequeued.

DESCRIPTION

The ESCSI_DEQ_FROM_TAIL_FAST macro dequeues a list element from the head or tail of a
given linked list, when it is known that the linked list is not empty; the macro does not check
whether the list is empty. This macro takes as an argument a pointer to the linked list header
__hdr and __elem, which is of type escsi_list_t *. The address of the list element dequeued
is placed in __elem.

RETURN VALUE

None

CONSTRAINTS

Do not use this macro when the list is empty.

EXAMPLE

escsi_list_t elem;
escsi_list_t header;
ESCSI_DEQ_FROM_TAIL_FAST(&header, &elem);

SEE ALSO

ESCSI_DEQ_FROM_TAIL(9G)
escsi_devt_to_lun(9F)

NAME

escsi_devt_to_lun - Convert a dev_t to a LUN pointer.

SYNOPSIS

escsi_lun_t *escsi_devt_to_lun(
    dev_t dev,
    int dev_type
);

PARAMETERS

dev The dev_t of the LUN being looked up,

dev_type D_BLK or D_CHR

DESCRIPTION

This SCSI Services public routine looks up the dev_t and returns the corresponding LUN structure if found. The dev_t can map to a legacy lun or a new LUN.

This routine is invoked in the context of a driver drv_ops_t entry point to convert.

RETURN VALUE

On successful return, this routine returns a pointer to a escsi_lun structure. On error, returns NULL.

CONSTRAINTS

None

SEE ALSO
NAME

escsi_dma32_map - Perform DMA mapping for a 32-bit only DMA capable card

SYNOPSIS

int escsi_dma32_map(
    struct buf *bp,
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_mem_handle_t *mem_handle,
    int num_iovecs,
    escsi_dma_parms dma_parms
);

PARAMETERS

bp Pointer to the buf structure.
isc Pointer to the isc structure of the controller.
dma_handle Specifies the DMA handle to be used during DMA unmap operation.
mem_handle Specifies the memory handle used to allocate 32-bit memory capable of direct memory access.
num_iovecs Specifies the number of I/O vectors allocated by the interface driver.
dma_parms Specifies the DMA control structure associated with bp.

DESCRIPTION

This function allocates the required 32-bit DMA capable data buffers that are CACHELINE aligned and maps them. The function does not support the 4GB split or the page alignment like the regular DMA map and unmap services. This service is to support existing drivers that can handle only 32-bit DMA. The existing alignment considerations are preserved.

The interface driver must allocate the scatter gather list and pass its pointer in the dma_parms object.

RETURN VALUE

ESUCCESS Mapping successful.
ENOMEM The DMA mapping failed or memory allocation for buflets failed.
ENOSPC The I/O vector array allocated by the caller is not sufficient to hold the DMA mappings.

CONSTRAINTS

The interface driver must not modify the dma_parms structure after calling escsi_dma32_map. This function can be called with spinlocks held.

EXAMPLE

    /* allocate dma_parms structure */
    dma_parms = (escsi_dma_parms_t *)kmem_arena_alloc(...);
    /* allocate iovec array */
    dma_parms->iovec = (struct iovec *)kmem_arena_alloc(...);
    /* perform DMA mapping */
    status = escsi_dma32_map(bp, isc, if_isc->dma_handle,
                        if_isc->mem_handle, num_iovecs, dma_parms);
SEE ALSO

buf(9S), escsi_dma32_unmap(9F)
NAME

escsi_dma32_unmap - Unmap the 32-bit DMA mapping and free resources

SYNOPSIS

void escsi_dma32_unmap(
    struct buf *bp,
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_mem_handle_t *mem_handle,
    escsi_dma_parms dma_parms
);

PARAMETERS

bp Pointer to the buf structure.
isc Pointer to the isc structure of the controller.
dma_handle Specifies the DMA handle used during the previous DMA map operation.
mem_handle Specifies the memory handle used to allocate 32-bit memory capable of direct
               memory access.
dma_parms Specifies the DMA control structure associated with bp.

DESCRIPTION

This function is called by interface driver to unmap a 32-bit DMA mapping for an I/O buffer.
This function copies the DMA'd data to the original data buffers if the I/O is a read. It then unmaps
the DMA mappings and deallocates the 32-bit DMA capable data buffers. The dma_parms
structure can be reused by the interface driver for another I/O only after this function returns.

RETURN VALUE

None

CONSTRAINTS

This function can be called with spinlocks held.

EXAMPLE

/ * unmap DMA mapping for the I/O */
escsi_dma32_unmap(bp, isc, if_isc->dma_handle, if_isc->mem_handle,
                   dma_parms);

SEE ALSO

buf(9F), escsi_dma32_map(9F)
NAME
escsi_dma_map - Perform DMA mapping for specified I/O buffer

SYNOPSIS

int escsi_dma_map(
  struct buf *bp,
  struct isc_table_type *isc,
  void *dma_handle,
  wsio_mem_handle_t *mem_handle,
  escsi_dma_req_flags_t dma_req_flags,
  uint32_t max_sgle_sz,
  int num_iovecs,
  escsi_dma_parms_t *dma_parms
);

PARAMETERS

bp       Pointer to the buf structure.
isc      Pointer to the isc structure of the controller.
dma_handle Specifies the DMA handle to be used during DMA map operation.
dma_req_flags   DMA request flags.
max_sgle_sz     Maximum number of bytes that can be mapped in single scatter/gather list
element.
num_iovecs    Number of I/O vector elements allocated.
dma_parms     DMA control object associated with the bp.

DESCRIPTION

This function is called by interface driver to perform DMA mapping for an I/O buffer. It can
handle merged I/Os, alignment restrictions, and the 4GB split requirement from interface driver.
The interface driver must not modify the dma_parms structure until the escsi_dma_unmap
routine is called. Before calling this function, the interface driver must allocate the required
number of iovec structures and pass them through the iovec_array field in dma_parms
structure. The number of iovec structures allocated is passed in num_iovecs. The caller must
allocate a DMA handle by calling wsio_allocate_dma_handle and pass it as the dma_handle
to this call.

The flag values allowed for dma_req_flags are:

DMA_REQ_DEFAULT_ALIGN The default alignment policy. The default policy is to only
                        align read requests on a cacheline boundary; write requests
                        are not aligned. Do not use the default alignment if the
                        interface driver required ifsw->if_beg_align and
                        ifsw->if_end_align in HP-UX version prior to 11i v3.

DMA_REQ_CACHE_ALIGN Indicates that the beginning and ending address must be
                        cacheline aligned for all I/Os (reads and writes).

DMA_REQ_PAGE_ALIGN Indicates that the beginning and ending address must be page
                        aligned for all I/Os (reads and writes).

DMA_REQ_IOVEC_SPLIT4GB Indicates that an I/O vector that straddles the 4GB address
                        space be split.
Alignment handling is checked only for unmerged requests because merged buf structures are always aligned. If default alignment is requested (no alignment bits set are in the dma_req_flags), only read I/Os are aligned on a cacheline boundary.

**RETURN VALUE**

ESUCCESS Mapping successful.
ENOMEM The DMA mapping failed or memory allocation for buflets failed.
ENOSPC The I/O vector array allocated by the caller is not sufficient to hold the DMA mappings.

**CONSTRAINTS**

This function can be called with spinlocks held.

**EXAMPLE**

```c
/* allocate dma_parms structure */
dma_parms = (escsi_dma_parms_t *)kmem_arena_alloc(...);
/* allocate iovec array */
dma_parms->iovec = (struct iovec *)kmem_arena_alloc(...);
/* perform DMA mapping */
status = escsi_dma_map(bp, isc, if_isc->dma_handle, 
                     DMA_REQ_CACHE_ALIGN, MAX_BYTES_PER_ SGLE, 
                     num_iovecs, dma_parms);
```

**SEE ALSO**

buf(9F), escsi_dma_unmap(9F), wsio_allocate_dma_handle(9F)
**escsi_dma_parms_t(9S)**

**NAME**

escsi_dma_parms_t - Structure that contains information that SCSI Services uses to perform DMA map and unmap operations.

**SYNOPSIS**

```c
#include <io/escsi_services.h>
```

**DESCRIPTION**

The `escsi_dma_parms_t` structure contains information that SCSI Services uses to perform DMA map and unmap operations for the I/O and by the interface driver to get the DMA address for programming the HBA to perform DMA. The `escsi_dma_parms_t` structure contains stateful information on the DMA mappings for an I/O, and is associated with a single I/O at any time. Do not reuse the structure for another I/O until the DMA mapping of the current I/O its associated with has been unmapped. The interface driver allocates the `escsi_dma_parms_t` structure and the scatter/gather list (S/G list) for each I/O.

**STRUCTURE MEMBERS**

```c
typedef struct {
    struct iovec    *iovec_array;
    int                mapped_iovecs;
    escsi_dma_flags_t flags;
    caddr_t            buflet_beg;
    caddr_t            buflet_end;
    char               **dma32_buf_array;
} escsi_dma_parms_t;
```

- **iovec_array**  
  S/G list. On input, the interface driver passes the pointer to the allocated S/G list. On return, the `escsi_dma_map` service initializes the S/G list with the DMA mappings for the I/O.

- **mapped_iovecs**  
  The number of S/G list elements in the `iovec_array` that contain DMA mappings.

- **flags**  
  Reserved for use by the `escsi_dma_map` and `unmap` services.

- **buflet_beg**  
  Pointer to the buflet managing the alignment of the start address. Reserved for use by the `escsi_dma_map` and `unmap` services.

- **buflet_end**  
  Pointer to the buflet managing the alignment of the end address. Reserved for use by the `escsi_dma_map` and `unmap` services.

- **dma32_buf_array**  
  Reserved for use by the SCSI Services for 32-bit DMA setup.

**SEE ALSO**
escsi_dma_req_flags_t(9S)

NAME

escsi_dma_req_flags_t - The DMA request flag enumeration.

SYNOPSIS

#include <sys/escsi_services.h>

DESCRIPTION

DMA request flags are set by the caller (interface driver) to request the alignment type and I/O vector split requirements.

STRUCTURE MEMBERS

typedef enum {
    DMA_REQ_DEFAULT_ALIGN  = 0x00,
    DMA_REQ_CACHE_ALIGN    = 0x01,
    DMA_REQ_PAGE_ALIGN     = 0x02,
    DMA_REQ_IOVEC_SPLIT4GB = 0x04
} escsi_dma_req_flags_t;

DMA_REQ_DEFAULT_ALIGN

Default alignment policy. The default policy is to only align read requests on a cacheline boundary. Write requests are not aligned.

CAUTION: Do not use the default alignment if the interface driver required ifsw->if_beg_align and ifsw->if_end_align in HP-UX versions prior to 11i v3.

DMA_REQ_CACHE_ALIGN

Indicates that the beginning and ending address must be cacheline aligned for all I/Os (reads and writes).

DMA_REQ_PAGE_ALIGN

Indicates that the beginning and ending address must be page aligned for all I/Os (reads and writes).

DMA_REQ_IOVEC_SPLIT4GB

Indicates that an I/O vector that straddles the 4GB or its multiple address boundary space be split.

EXAMPLE

None

SEE ALSO

898 SCSI Reference Pages
NAME

escsi_dma_unmap - Unmap the DMA mapping for an I/O buffer

SYNOPSIS

void escsi_dma_unmap(
    struct buf *bp,
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_mem_handle_t *mem_handle,
    escsi_dma_req_flags_t dma_req_flags,
    escsi_dma_parms_t *dma_parms
);

PARAMETERS

bp Pointer to the buf structure.
isc Pointer to isc structure of the controller.
dma_handle The DMA handle to be used during DMA unmap operation.
dma_req_flags DMA request flags.
dma_parms Pointer to the DMA control structure associated with the bp.

DESCRIPTION

This function is called by interface driver to unmap the DMA mapping for an I/O buffer. This function manages the data transferred by buflets and frees them. It performs dma_sync if the I/O is a read.

The flag values allowed for dma_req_flags are:

DMA_REQ_DEFAULT_ALIGN The default alignment policy. The default policy is to only align read requests on a cacheline boundary; write requests are not aligned. Do not use the default alignment if the interface driver required ifsw->if_beg_align and ifsw->if_end_align in HP-UX version prior to 11i v3.

DMA_REQ_CACHE_ALIGN Indicates that the beginning and ending address must be cacheline aligned for all I/Os (reads and writes).

DMA_REQ_PAGE_ALIGN Indicates that the beginning and ending address must be page aligned for all I/Os (reads and writes).

DMA_REQ_IOVEC_SPLIT4GB Indicates that an I/O vector that straddles the 4 GB address space be split.

RETURN VALUE

None

CONSTRAINTS

This function can be called with spinlocks held.

EXAMPLE

/* unmap DMA mapping for the I/O */
escsi_dma_unmap(bp, isc, if ISC->dma_handle,
    DMA_REQ_CACHE_ALIGN, dma_parms);
SEE ALSO

`buf(9F), escsi_dma_map(9F)`
escsi_drv_cb(9F)

NAME

escsi_drv_cb – Default entry point for class drivers

SYNOPSIS

int escsi_drv_cb(
    drv_cb_opcode_t cb_opcode,
    uintptr_t *data
);

PARAMETERS

cb_opcode Either CB_GET_OPTS or CB_DEV_2_NODE.
data Pointer to the opcode-specific data to return.

DESCRIPTION

If the opcode is CB_GET_OPTS, the function returns the device-specific options. If the opcode is CB_DEV_2_NODE, the function returns a pointer to the LUN I/O node corresponding to the devt.

RETURN VALUES

GIO_SUCCESS The opcode was successfully processed.
GIO_ERROR The opcode processing failed.

CONSTRAINTS

This function must be used as the drv_ops_t->d_drv_cb entry point of the class driver.

SEE ALSO

901
NAME
escsi_enq_active_lpt - Queue a lunpath onto the active lunpath queue

SYNOPSIS

void escsi_enq_active_lpt(
    escsi_lun_t *lun,
    escsi_lpt_t *lpt
);

PARAMETERS

lun    Pointer to the lun structure.

lpt    Pointer to the lunpath element to queue to the LUN active lunpath queue.

DESCRIPTION

This service queues a lunpath in the LUN active path queue.

RETURN VALUES

None

CONSTRAINTS

The caller must hold a lun lock.
The lpt that is requested to be dequeued must be an active lunpath and not already on the LUN active path queue.

EXAMPLE

void mydriver_dd_aen(void *lpt_hdl, escsi_async_evt_t event, intptr_t ev_arg)
{
    switch (event) {
    case ESCSI_LPT_ONLINE :
        escsi_lpt_online(lpt);

        /* Dequeue path from active path queue */
        ESCSI_LUN_LOCK(lun);

        /* Dequeue offline path from lun active path queue */
        if (escsi_is_on_q(&lun->active_path_q,
            (escsi_list_t *)lpt) == ESCSI_FALSE) {
            escsi_enq_active_lpt(lun, lpt);
        }

        /* Class driver specific lunpath activation action */
       ..
        ..

        /* Compute path counts for the LUN */
        escsi_lun_compute_path_cnt(lun);
        ESCSI_LUN_UNLOCK(lun);
        break;
    ..
    ..
    }
return;
}

**SEE ALSO**

escsi_deq_active_lpt(9F), escsi_is_on_q(9F), escsi_lpt_offline(9F), lun_compute_path_cnt(9F)
ESCSI ENQ AFTER(9G)

NAME
ESCSI_ENQ_AFTER - Macro to queue a linked list element after the specified list element.

SYNOPSIS

ESCSI_ENQ_AFTER( escsi_list_t *__prev_elem,
                escsi_list_t *__elem)

PARAMETERS

__prev_elem Pointer to previous list element.
__elem Pointer to list element to be queued.

DESCRIPTION

The ESCSI_ENQ_AFTER macro queues a given element __elem after the __prev_elem element in a linked list.

RETURN VALUE

None

CONSTRAINTS

None

EXAMPLE

escsi_list_t pre_elem;
escsi_list_t elem;
ESCSI_ENQ_AFTER(&prev_elem, &elem);

SEE ALSO

ESCSI_ENQ BEFORE(9G)
ESCSI_ENQ_AT_HEAD(9G)

NAME
ESCSI_ENQ_AT_HEAD - Macro to queue a linked list element at the head of the queue.

SYNOPSIS
ESCSI_ENQ_AT_HEAD( escsi_list_t *__hdr,
                   escsi_list_t * __elem)

PARAMETERS
__hdr    Pointer to the list header.
__elem   Pointer to the list element.

DESCRIPTION
The ESCSI_ENQ_AT_HEAD macro queues a given linked list element at the head of the queue.

RETURN VALUE
None

CONSTRAINTS
None

EXAMPLE
escci_list_t queue_header;
escci_list_t elem;
ESCSI_ENQ_AT_HEAD(&queue_header, &elem);

SEE ALSO
ESCSI_ENQ_AT_TAIL(9G)
ESCSI_ENQ_AT_TAIL(9G)

NAME
ESCSI_ENQ_AT_TAIL - Macro to queue a linked list element at the tail of the queue.

SYNOPSIS

ESCSI_ENQ_AT_TAIL( escsi_list_t *__hdr,
          escsi_list_t * __elem)

PARAMETERS

__hdr   Pointer to the list header.
__elem   Pointer to the list element.

DESCRIPTION

The ESCSI_ENQ_AT_TAIL macro queues a given linked list element at the tail of the queue.

RETURN VALUE

None

CONSTRAINTS

None

EXAMPLE

escsi_list_t queue_header;
escsi_list_t elem;
ESCSI_ENQ_AT_TAIL(&queue_header, &elem);

SEE ALSO

ESCSI_ENQ_AT_HEAD(9G)
ESCSI_ENQ_BEFORE(9G)

NAME
ESCSI_ENQ_BEFORE – Macro to queue a linked list element before the specified list element.

SYNOPSIS
ESCSI_ENQ_BEFORE( escsi_list_t *__next_elem, escsi_list_t *__elem)

PARAMETERS
__next_elem Pointer to next list element.
__elem Pointer to list element to be queued.

DESCRIPTION
The ESCSI_ENQ_BEFORE macro queues a given __elem element before the __next_elem element in a linked list.

RETURN VALUE
None

CONSTRAINTS
None

EXAMPLE
escsi_list_t next_elem;
escsi_list_t elem;

ESCSI_ENQ_BEFORE(&next_elem, &elem);

SEE ALSO
ESCSI_ENQ_AFTER(9G)
ESCSI_ENQ_BP_AT_HEAD(9G)

NAME

ESCSI_ENQ_BP_AT_HEAD - Macro to queue a buf structure element at the head of a doubly linked list.

SYNOPSIS

ESCSI_ENQ_BP_AT_HEAD(__qh, __bp)

PARAMETERS

__qh Pointer to the head of the queue.
__bp Pointer to a buf structure element in the queue.

DESCRIPTION

The ESCSI_ENQ_BP_AT_HEAD macro queues a buf structure at the head of a doubly linked list.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

struct buf *buf_q; /* Head of bp queue */
struct buf *bp;

/*
 * Queue the bp element at the head of the queue.
 */
ESCSI_ENQ_BP_AT_HEAD(&buf_q, bp);

SEE ALSO

ESCSI_ENQ_BP_AT_TAIL(9G)
ESCSI_ENQ_BP_AT_TAIL(9G)

NAME

ESCSI_ENQ_BP_AT_TAIL - Macro to queue a buf structure element at the tail of a doubly linked list.

SYNOPSIS

ESCSI_ENQ_BP_AT_TAIL(__qh, __bp)

PARAMETERS

__qh  Pointer to the head of the queue.
__bp  Pointer to a buf structure element in the queue.

DESCRIPTION

The ESCSI_ENQ_BP_AT_TAIL macro queues a buf structure at the tail of a doubly linked list.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

struct buf  *buf_q;     /* Head of bp queue */
struct buf  *bp;

/*
 * Queue the bp element at the tail of the queue.
 */
ESCSI_ENQ_BP_AT_TAIL(&buf_q, bp);

SEE ALSO

ESCSI_ENQ_BP_AT_HEAD(9G)
escsi_get_addr(9F)

NAME
escsi_get_addr - Get a escsi_addr_t structure for the specified lunpath

SYNOPSIS
int escsi_get_addr(
    dev_t ldev,
    escsi_fetch_hint_t hint,
    sioc_lpt_type_t type,
    escsi_addr_t *cur_addr,
    escsi_addr_t *next_addr
);

PARAMETERS
ldev Specifies the local devt of the LUN.
hint Fetch hint specifying the starting point of the search. Supported fetch hints are
    ESCSI_GET_FIRST and ESCSI_GET_NEXT.
type Specifies the type of lunpath being searched. The type must equal to any one of
    the following: SIOC_BOOT_LPTS, SIOC_DUMP_LPTS, and SIOC_ALL_LPTS.
cur_addr Pointer to a escsi_addr_t structure in which to search for the appropriate
    lunpath.
next_addr Pointer to a escsi_addr_t structure in which to return the address information
    for the found lunpath.

DESCRIPTION
The escsi_get_addr function gets the escsi_addr_t instance address for a specified lunpath.
The caller can specify the lunpath using local dev_t or ask for the next lunpath providing an
escsi_addr_t. The lunpaths being searched can be either boot lunpaths only or dump lunpaths
only, or both. The type of search needs to be specified accordingly as SIOC_BOOT_LPTS,
SIOC_DUMP_LPTS, or SIOC_ALL_LPTS.

RETURN VALUE
ENXIO Invalid devt.
ENODEV No suitable lunpath found, or reached end of path queue.
EINVAL Invalid argument passed in.
ESUCCESS Success and next_addr contains the c-t-l information of the found lunpath.

CONSTRAINTS
This function can only be called in blocking context.
No spinlocks can be held by the caller.

EXAMPLE
escsi_addr_t    first;
escsi_addr_t    next;
int             errno = ESUCCESS;
errno = escsi_get_addr(ldev, ESCSI_GET_FIRST, SIOC_ALL_LPTS,
    NULL, &first);
if (errno != ESUCCESS) {
    process errno;
    return;
}
next = first;
do {
    process the lpt in next;
    errno = escsi_get_addr(ldev, ESCSI_GET_NEXT,
                         SIOC_ALL_LPTS, &next, &next);
    if (errno != 0) {
        break;
    }
} while (1);

if (errno != ESUCCESS) {
    process errno;
    return;
}

SEE ALSO

escsi_addr_t(9S), escsi_fetch_hint_t(9S)
NAME
escsi_get_any_path - Get an active or standby path to the LUN.

SYNOPSIS

```c
escsi_lpt_t escsi_get_any_path(
    escsi_lun_t *lun,
    escsi_lpt_t *clpt
);
```

PARAMETERS

lun Pointer to a escsi_lun structure.
clpt Specifies the current lunpath pointer. This can be NULL. This parameter specifies where to start the search for the active or standby path.

DESCRIPTION

This routine obtains the first active path in the lunpath queue. If there is no active path, the routine attempts to find a standby path. Load balancing policy of the LUN is not applied to select the path.

RETURN VALUE

lpt Pointer to escsi_lpt structure if path is found.
NULL No lunpath was found.

CONSTRAINTS

The caller must hold a LUN lock.
This function does not sleep.

EXAMPLE

```c
int
mydriver_open(dev_t dev, int oflags, intptr_t dummy, int mode)
{
    escsi_lun_t     *lun;
    escsi_lpt_t     *lpt;
    int             errno;

    /* Lookup lun dev_t and open the lun */
    lun = escsi_devt_to_lun(dev, dev_type);
    errno = escsi_lun_open(lun, dev, D_CHR);

    /* Select any lunpath under the LUN */
    lpt = escsi_get_any_path(lun, NULL);
    .
    .
    .
    return errno;
}
```

SEE ALSO

escsi_get_best_lpt(9F), escsi_get_path(9F), escsi_lpt_t (9S), escsi_lun_t (9S)
NAME
escsi_get_best_lpt - Select the lunpath with the least I/O traffic load.

SYNOPSIS

escsi_lpt_t *escsi_get_best_lpt(
   escsi_lun_t *lun
);

PARAMETERS

lun Pointer to an escsi_lun structure.

DESCRIPTION

This routine obtains the lunpath with the least I/O load.

RETURN VALUES

lpt Pointer to an escsi_lpt structure if the path is found.
NULL No lunpath was found.

CONSTRAINTS

The caller must hold a LUN lock.
The function does not sleep.

EXAMPLE

int
mydriver_open(dev_t dev, int oflags, intptr_t dummy, int mode)
{
    escsi_lun_t     *lun;
    escsi_lpt_t     *lpt;
    int             errno;

    /* Lookup lun dev_t and open the lun */
    lun = escsi_devt_to_lun(dev, dev_type);
    errno = escsi_lun_open(lun, dev, D_CHR);

    /* Select lpt with least IO load */
    lpt = escsi_get_best_lpt(lun);

    .
    .
    .
    return errno;
}

SEE ALSO

escsi_get_any_path(9F), escsi_get_path(9F), escsi_lpt_t(9S), escsi_lun_t(9S)
**NAME**

escsi_get_class_inst - Get the class and instance number given eSCSI information or eSCSI address.

**SYNOPSIS**

```c
int escsi_get_class_inst(
    escsi_info_t *info,
    escsi_addr_t *addr,
    escsi_type_t type,
    char *class,
    int *inst
);
```

**PARAMETERS**

- **info** If not null, the service returns the class and instance number of the I/O node corresponding to the `info` and `type` parameters.
- **addr** If the `info` parameter is NULL, the caller can pass a pointer to `escsi_addr_t`. The service returns the class and instance number of the I/O node corresponding to the `addr` and `type` parameters.
- **type** LUN or lunpath.
- **class** On return, the class value.
- **inst** On return, the instance value.

**DESCRIPTION**

This routine returns the class and instance number for a SCSI object identified by one of the following parameter combinations:

- A pointer to `escsi_info_t` and an `escsi_type_t` object type
- A pointer to `escsi_addr_t` and an `escsi_type_t` object type

**RETURN VALUES**

- **ESUCCESS** Class and instance are successfully obtained.
- **EINVAL** Invalid identification of the SCSI object.

**CONSTRAINTS**

If the caller passes `info` as NULL, the `addr` and `type` parameters must be valid.

**EXAMPLE**

```c
escsi_lun_t *lun;
char   class[512];
int    inst;

error = escsi_get_class_inst((escsi_info_t *)lun,
                            NULL, ESCSI_LUN, class,
                            &inst);
```

**SEE ALSO**

`escsi_get_any_path(9F), escsi_get_path(9F), escsi_lpt_t(9S), escsi_lun_t(9S)`
escsi_get_inq_serial(9F)

NAME

escsi_get_inq_serial - Retrieve the Inquiry data for a LUN.

SYNOPSIS

#include <sys/escsi_services.h>

int escsi_get_inq_serial(
    escsi_inq_serial_t *serialinfo
);

PARAMETERS

serialinfo Pointer to a escsi_inq_serial_t structure, which is defined in the
            escsi_services.h header file.

DESCRIPTION

The escsi_get_inq_serial function retrieves the inquiry data for the LUN specified in the
inq_data field of the escsi_inq_serial_t structure. Before calling this function, initialize
the addr field of the input structure to the correct lunpath address.

RETURN VALUES

ESUCCESS Inquiry data retrieved successfully.
EINVAL Failure. Possible reasons include:
    • The escsi_get_inq_serial function was called with a lock held.
    • The escsi_addt_t field in the escsi_inq_serial_t structure is invalid.

CONSTRAINTS

None

SEE ALSO

escsi_apsw_reg(9F), escsi_cdbinfo_init(9F), escsi_cmdx_ext(9F), escsi_inq_serial_t(9S)
ESCSI_GET_LEG_BUS_ID(9G)

NAME
ESCSI_GET_LEG_BUS_ID - Macro to extract legacy bus identifier from a legacy dev_t

SYNOPSIS
ESCSI_GET_LEG_BUS_ID(dev_t _dev)

PARAMETERS
_dev Legacy dev_t.

DESCRIPTION
The ESCSI_GET_LEG_BUS_ID macro returns the legacy bus identifier from the legacy dev_t passed as parameter.

RETURN VALUE
Bus identifier.

EXAMPLE
lpt = ESCSI_GET_LEG_BUS_ID(&activeq_hdr);

CONSTRAINTS
The _dev must be a legacy dev_t. That means it must correspond to the format of dev_t used in releases prior to HP-UX 11i v3.

SEE ALSO
ESCSI_GET_LEG_LUN_ID(9G), ESCSI_GET_LEG_TGT_ID(9G)
ESCSI_GET_LEG_LUN_ID(9G)

NAME
ESCSI_GET_LEG_LUN_ID - Macro to extract legacy LUN identifier from a legacy dev_t

SYNOPSIS
ESCSI_GET_LEG_LUN_ID(dev_t _dev)

PARAMETERS
_dev    Legacy dev_t.

DESCRIPTION
The ESCSI_GET_LEG_LUN_ID macro returns the legacy LUN identifier from the legacy dev_t passed as parameter.

RETURN VALUE
Bus identifier.

EXAMPLE
lpt = ESCSI_GET_LEG_LUN_ID(&activeq_hdr);

CONSTRAINTS
The _dev must be a legacy dev_t. That means it must correspond to the format of dev_t used in releases prior to HP-UX 11i v3.

SEE ALSO
ESCSI_GET_LEG_BUS_ID(9G), ESCSI_GET_LEG_TGT_ID(9G)
ESCSI_GET_LEG_TGT_ID(9G)

NAME
ESCSI_GET_LEG_TGT_ID - Macro to extract legacy target identifier from a legacy dev_t

SYNOPSIS
ESCSI_GET_LEG_TGT_ID(dev_t _dev)

PARAMETERS
_dev    Legacy dev_t.

DESCRIPTION
The ESCSI_GET_LEG_TGT_ID macro returns the legacy target identifier from the legacy dev_t passed as parameter.

RETURN VALUE
Bus identifier.

EXAMPLE
lpt = ESCSI_GET_LEG_TGT_ID(&activeq_hdr);

CONSTRAINTS
The _dev must be a legacy dev_t. That means it must correspond to the format of dev_t used in releases prior to HP-UX 11i v3.

SEE ALSO
ESCSI_GET_LEG_BUS_ID(9G), ESCSI_GET_LEG_LUN_ID(9G)
ESCSI_GET_PARENT(9G)

NAME
ESCSI_GET_PARENT - Macro to get the parent pointer of a linked list element

SYNOPSIS
ESCSI_GET_PARENT( escsi_list_t *__elem)

PARAMETERS
__elem Pointer to the list element.

DESCRIPTION
The ESCSI_GET_PARENT macro returns pointer to the parent structure of given list element.

RETURN VALUE
A pointer to the parent structure.

EXAMPLE
See ESCSI_NEXT(9G) example.

CONSTRAINTS
None

SEE ALSO
ESCSI_NEXT(9G), ESCSI_SET_PARENT(9G)
NAME

escsi_get_options - Get the device-specific options for a specified new LUN dev_t.

SYNOPSIS

int escsi_get_options(
    dev_t dev,
    uint64_t *options
);

PARAMETERS

dev A new LUN dev_t.

options Device-specific options for the specified dev_t.

DESCRIPTION

This routine returns the device-specific options registered for the specified dev_t.

RETURN VALUES

ESUCCESS Device-specific options were found.

ENXIO No LUN with the corresponding dev_t was found.

CONSTRAINTS

Do not hold a spinlock around this routine.

EXAMPLE

SEE ALSO
escsi_get_path(9F)

NAME

escsi_get_path - Select an active path for a given I/O.

SYNOPSIS

escsi_lpt_t *escsi_get_path(
    escsi_lun_t *lun,
    struct buf *bp
);

PARAMETERS

lun Pointer to a escsi_lun structure.
bp Pointer to a buf structure.

DESCRIPTION

The escsi_get_path routine selects a path for a specified I/O and LUN, based on the type of the I/O and the load balancing policy on the LUN. Class drivers can use this service in a context where a new path has to be reselected to issue an I/O request. The routine uses the following algorithm to select a path:

- If I/O is an MPAI I/O, a MPAI path is selected.
- If load balancing policy on the LUN is round robin, the next path in the path queue is selected for issuing the I/O.
- If the load balancing policy on the LUN is least command load, an active path that has the least number of outstanding I/Os is selected for issuing the I/Os.
- If the load balancing policy is path lockdown, the path cached by the class driver in lun->next_lpt is selected for issuing the I/Os.

This function assumes that the class driver initializes the escsi_lun_t->next_lpt to a path in the active path set. If the path pointed to by escsi_lun_t->next_lpt goes offline or to a standby state, the class driver must update this field with another path from active path set. A NULL value for the escsi_lun_t->next_lpt implies there are no active paths for the LUN. For path lockdown policy, the escsi_lun_t->next_lpt must be set to the lock down path.

RETURN VALUE

On successful return, the routine returns a pointer to a escsi_lpt structure. On error, returns a NULL pointer.

CONSTRAINTS

The caller must hold a LUN lock.
This function does not sleep.

SEE ALSO

escsi_get_any_path(9F), escsi_get_best_lpt(9F)
NAME
escsi_global_t - SCSI subsystem global data structure

SYNOPSIS
#include <io/escsi_stack.h>

STRUCTURE MEMBERS
info  Pointer to an escsi_info_t structure. This field often identifies a SCSI object type. The info->type field is set to ESCSI_GLOBAL.
arena A pointer to the memory arena for dynamically allocating memory in the SCSI global memory arena.

DESCRIPTION
The escsi_global variable holds SCSI subsystem global information. Class drivers access the previous fields.

SEE ALSO
escsi_info_t(9S), SIOC_ATTR(9G)
ESCSI_HOLD(9G)

NAME

ESCSI_HOLD - Macro to hold a SCSI object to prevent its deletion.

SYNOPSIS

ESCSI_HOLD(__obj, __owns_lock)

PARAMETERS

__obj Pointer to the escsi_info_t structure of the SCSI object.
__owns_lock Boolean to determine whether the macro must hold and release the SCSI object lock while performing the hold operation.

DESCRIPTION

The ESCSI_HOLD macro is a non-blocking synchronization mechanism for protecting a SCSI object from deletion. Several operations can concurrently invoke ESCSI_HOLD on the same object. If the object lock is not already held while invoking ESCSI_HOLD, the caller must set the __owns_lock parameter to ESCSI_FALSE. When an operation does not need to protect the SCSI object from deletion, it invokes ESCSI_RELE. The SCSI object can only be deleted when the last ESCSI_RELE is invoked.

RETURN VALUES

None

CONSTRAINTS

If the SCSI object lock is not already held while invoking ESCSI_HOLD, the caller must set the __owns_lock parameter to ESCSI_FALSE.

ESCSI_RELE must be invoked after invoking ESCSI_HOLD.

EXAMPLE

Following is an example of the ESCSI_HOLD and ESCSI_RELE macros in the context of a legacy class driver ioctl entry point.

```c
int mydriver_leg_ioctl(dev_t dev, int cmd, caddr_t udata_ptr, int flags)
{
    escsi_lpt_t     *lpt;
    escsi_lun_t     *lun;
    escsi_leg_lun_t *leg_lun;
    int             errno;

    leg_lun = escsi_leg_dev_lookup(dev);
    if (leg_lun == NULL) {
        return ENODEV;
    }
    /* Look up lpt from leg_lun->lpt and hold lpt */
    ESCSI_LEG_LUN_LOCK(leg_lun);
    lpt = leg_lun->lpt;
    lun = leg_lun->lun;
    if ((lpt == NULL) || (lun == NULL)) {
        return ENODEV;
    }
    /* Holding the lpt prevents its deletion. */
```
ESCSI_HOLD(lpt, ESCSI_FALSE);
ESCSI_LEG_LUN_UNLOCK(leg_lun);

errno = escsi_ioctl(dev, lpt, cmd, udata_ptr, flags);

ESCSI_RELE(lpt, ESCSI_FALSE);

return errno;

}
NAME
escsi_if_aen - Asynchronous event notification to SCSI Services

SYNOPSIS
int escsi_if_aen(
    escsi_aen_t *aen
);

PARAMETERS
aen  Pointer to asynchronous event notification structure.

DESCRIPTION
The escsi_if_aen function performs asynchronous event notification to SCSI Services. The
aen structure contains the event id and escsi_addr_t instance address of the target path for
which the event is being sent. The AEN event IDs that can be sent using this interface are:

- **ESCSI_TP_OFFLINE**: Notify SCSI Services about a target path going offline. Mandatory
  for all interface drivers.
- **ESCSI_TP_ONLINE**: Notify SCSI Services about a target path coming online. Mandatory
  for all interface drivers.
- **ESCSI_IOBJ_CPU_REG**: Bind an I/O object to a CPU. Mandatory for interface drivers
  supporting line-based or transaction-based interrupts.
- **ESCSI_IF_IOBJ_REG**: Bind an interface specific object to the I/O object. Must be used
  by drivers that wish to associate an interface-specific object with the I/O object.
- **ESCSI_PORT_ID_CHANGE**: Notify SCSI Services about a target port ID change. The SCSI
  Services require that the target path have a port name and a node name.

The **ESCSI_TP_OFFLINE**, **ESCSI_TP_ONLINE** and **ESCSI_PORT_ID_CHANGE** events can be
sent only for target paths that have been successfully registered with SCSI Services previously.

RETURN VALUE
- **ESUCCESS**: Asynchronous event notification succeeded.
- **EINVAL**: Invalid parameter passed

CONSTRAINTS
This function can be invoked in ICS context or in a blocking context.
No spinlock must be held by the caller.

EXAMPLE
1. /* send an offline event for the target path */
   escsi_aen_t  aen;
   aen->version = ESCSI_AEN_VER;
   aen->evt = ESCSI_TP_OFFLINE;
   aen->addr = iftgt_reg->addr;
   aen->evt_data = NULL;
   escsi_if_aen(&aen);
2. /* bind I/O object with CPU */
   escsi_aen_t aen;
   escsi_iobj_cpu_t iobj_cpu;

   .

   wsio_intr_get_assigned_cpu(glue->intr_object_handle, &cpu_id);
   aen.evt = ESCSI_IOBJ_CPU_REG;
   aen.version = ESCSI_AEN_VER;
   aen.addr.c = escsi_ifctrlr_reg_t->addr.c;
   aen.addr.t = aen.addr.l = 0;
   iobj_cpu.ioobj_id = 0;
   iobj_cpu.cpu_id = (uint64_t)cpu_id;
   aen.evt_data = (intptr_t)&iobj_cpu;
   escsi_if_aen(&aen);

SEE ALSO

escsi_aen_t(9S)
escsi_if_ctlr_prb_cbfn(9F)

NAME

escsi_if_ctlr_prb_cbfn - Controller probe completion callback function

SYNOPSIS

void escsi_if_ctlr_prb_cbfn(
    escsi_ifctrlr_reg_t *ifctlr_reg,
    int errno
);

PARAMETERS

ifctlr_reg Pointer to a controller registration data structure.
errno The errno value indicating completion status of probe.

DESCRIPTION

This function is called by interface driver to inform SCSI Services about completion of controller
probe activity. Interface driver calls back this function upon completion of the last target path
registration during the controller probe.

The values that can be passed by interface driver as errno value for probe completion status
are:
ESUCCESS Controller probe is successful.
EBUSY Transport or device is currently busy.
ENXIO Probe failed due to controller offline or some other error.
ENOMEM No memory resources to perform requested operation.

As part of probe completion, interface driver must also send an ESCSI_TP_OFFLINE event for
all targets that were registered earlier but not found in the current probe instance.

RETURN VALUE

None

CONSTRAINTS

This function can be called in ICS context or blocking context.
No spinlocks must be held by the caller.

EXAMPLE

xxx_ctlr_probe(void *if_ctlr)
{
    /* Perform transport specific probe to discover targets */
    ...
    for all targets discovered {
        /* Register target with SCSI Services */
        escsi_tgt_reg(...);
    }
} drv_tgt_reg_cbfn()
{
    /* Check target reg completion status. */
    ...
    /* If last target reg complete, inform SCSI Services. */
escsi_if_ctlr_prb_cbfn(ifctrl_reg, ESUCCESS);
}

SEE ALSO

escsi_ifctrl_reg(9S), escsi_tgt_reg(9F)
escsi_if_flags_t(9S)

NAME
escsi_if_flags_t - Flags set by interface drivers during the registration of the escsi_ifsw_t structure.

SYNOPSIS
#include <sys/escsi_services.h>

DESCRIPTION
Interface flags are set by the interface driver during registration of the escsi_ifsw_t structure. IF_B2_LIST is obsoleted because all drivers need to support B2_LIST. IF_NO_TAGS is obsoleted because the driver can indicate non-usage of tags by not using any of these tag flags and by setting if_max_tags to 0.

STRUCTURE MEMBERS
typedef enum {
    IF_IOBJ_TAGS             = 0x01,
    IF_TP_TAGS               = 0x02,
    IF_LPT_TAGS              = 0x04,
    IF_CTLR_TRACING          = 0x08
} escsi_if_flags_t;

IF_IOBJ_TAGS Indicate that the interface driver supports tagging per I/O object. This is the default tagging level.

IF_TP_TAGS Indicates that the interface driver supports tagging per target path.

IF_LPT_TAGS Indicates that the interface driver supports tagging per lunpath.

IF_CTLR_TRACING Indicates that the interface driver wants to use the tracing services. The interface driver must also specify the tracing buffer size (see escsi_ifctlr_attr_t(9S)).

SEE ALSO
escsi_ifctlr_attr_t(9S)
escsi_if_iobj_t(9S)

NAME
escsi_if_iobj_t - Parameter of the ESCSI_IF_OBJ_REG operation

SYNOPSIS
#include <io/escsi_services.h>

DESCRIPTION
The escsi_if_iobj_t data structure is used to bind an interface driver private I/O object handle with an I/O object. When the interface driver registers an SCSI HBA controller, it typically specifies the number of I/O objects to be allocated by SCSI Services. The interface driver must bind a private handle to each I/O object. This private handle allows the interface driver to determine the resources associated with the I/O object when processing I/Os. For that purpose, the interface driver must invoke the escsi_if_aen function with ESCSI_OBJ_REG as the operation and initialize the evt_data field of the escsi_aen_t structure with a pointer to a escsi_if_iobj_t data structure.

STRUCTURE MEMBERS
typedef struct {
    uint32_t          iobj_id;
    void             *if_iobj;
} escsi_if_iobj_t;

iobj_id   Index of the I/O object.
if_iobj   Pointer to an interface driver private handle for the I/O object.

SEE ALSO
escsi_aen_t(9S), escsi_if_aen(9F), escsi_iobj_get(9F), escsi_iobj_cpu_t(9S), escsi_iobj_t(9S)
NAME

escsi_ifctlr_attr_t - SCSI HBA controller attributes and statistics structure.

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

The escsi_ifctlr_attr_t structure contains attributes for a SCSI HBA controller. When an interface driver registers a SCSI HBA controller with SCSI Services, the driver provides a pointer to this structure.

STRUCTURE MEMBERS

typedef struct {
    escsi_name_t           iport_name;
    uint64_t                    port_id;
    char                         *protocol;
    char                         *driver_name;
    char                         *driver_desc;
    char                         *driver_rev;
    char                         *fw_rev;
    char                         *cur_speed;
    char                         *max_speed;
    uint32_t                    addnl_trc_rec;
    uint32_t                    num_iobj;
    escsi_ifctlr_capability_t   capability;
    escsi_ifctlr_type_t         type;
} escsi_ifctlr_attr_t;

iport_name Controller’s worldwide unique name. Can be modified by the driver.

port_id Controller’s port identifier. This is a SCSI transport protocol defined address for the initiator port. Can be modified by the driver.

protocol The SCSI transport protocol implemented by this controller. This attribute can take one of the following values: IDE, PARALLEL_SCSI, FIBRE_CHANNEL, ISCSI, SAS, or SRP. This field must be initialized with a non-NULL character. Cannot be modified by the driver.

driver_name Specifies the driver name. This field must be initialized with a non-NULL character. Cannot be modified by the driver.

driver_desc Specifies a descriptive name for the controller. This field must be initialized with a non-NULL character. Can be modified by the driver.

driver_rev Specifies the version of the interface driver managing this controller. This field must be initialized with a non-NULL character. Cannot be modified by the driver.

fw_rev Specifies the firmware version of the controller. This field must be initialized with a non-NULL character. Can be modified by the driver.

cur_speed Specifies the current negotiated or configured speed (throughput) at which the SCSI controller is operating. This field must be initialized with a non-NULL character. Can be modified by the driver.

max_speed Specifies the maximum speed (throughput) supported by the controller. This field must be initialized with a non-NULL character. Can be modified by the driver.
addnl_trc_rec  Specifies additional trace records when interface drivers intend to use unified tracing. Used in conjunction with the ifsw->if_flags IF_CTLR_TRACING flag.

num_iobj  Specifies the number of I/O objects.

capability  Specifies whether the SCSI controller has boot or dump IODC and EFI dump driver support on the current platform.

type  Specifies the controller type (logical or physical).

**CONSTRAINTS**

None

**SEE ALSO**

escsi_ctlr_reg(9F), escsi_ifctlr_reg_t(9S), escsi_ifctlr_stat_t(9S)
NAME

escsi_ifctlr_get - Gets interface controller handle for specified controller

SYNOPSIS

```
escsi_ifctlr_reg_t *escsi_ifctlr_get(
    escsi_addr_t *addr,
    escsi_fetch_hint_t hint,
    char *drv_name
);
```

PARAMETERS

- `addr`: Pointer to an eSCSI address structure.
- `hint`: Specifies the fetch hint.
- `drv_name`: Specifies the driver name string.

DESCRIPTION

This function can be used by interface drivers to get the interface specific registration handle for the controller structure. It returns the `ifctlr_reg` handle corresponding to the `escsi_addr_t` and the fetch hint supplied. The fetch hints allowed are:

- **ESCSI_GET_THIS**: Return the `ifctlr_reg` handle corresponding to the SCSI controller of the `addr.c` instance number.

- **ESCSI_GET_FIRST**: Return the `ifctlr_reg` handle corresponding to the first SCSI controller of the specified driver name found in the SCSI controller hash table.

- **ESCSI_GET_NEXT**: Return the `ifctlr_reg` handle corresponding to the next component SCSI controller after the SCSI controller component of `addr` for the specified driver name.

The following table shows the valid parameter combinations:

<table>
<thead>
<tr>
<th><code>hint</code></th>
<th><code>addr</code></th>
<th><code>drv_name</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ESCSI_GET_THIS</strong></td>
<td><code>addr.c</code></td>
<td>NULL</td>
</tr>
<tr>
<td><strong>ESCSI_GET_FIRST</strong></td>
<td>NULL</td>
<td>Driver name</td>
</tr>
<tr>
<td><strong>ESCSI_GET_NEXT</strong></td>
<td><code>addr.c</code> of the previously fetched element.</td>
<td>Driver name</td>
</tr>
</tbody>
</table>

The `ESCSI_GET_FIRST` and `ESCSI_GET_NEXT` hints refer to the order of the elements in the SCSI Services hash table, not the order of the element sorted by `addr address` values. The `drv_name` parameter is the driver name string and must be same as the one passed in `escsi_ifctlr_attr_t` during controller registration.

RETURN VALUE

If successful, the routine returns a pointer to the `escsi_ifctlr_reg_t` structure. If the requested controller element has not been registered, the routine returns NULL.

CONSTRAINTS

This function can be called in ICS context or blocking context.

The interface driver cannot hold spinlocks while calling this function.
EXAMPLE

escsi_addr_t addr;
.
.
/* Get ifctlr_reg in a loop for all controller
 * instances under the driver *
 */
drv_name = "td";
ifctlr_reg = escsi_ifctlr_get(NULL, ESCSI_GET_FIRST, drv_name);
while (ifctlr_reg != NULL) {
  .
  .
  .
  ifctlr_reg = escsi_ifctlr_get (ifctlr_reg->addr, ESCSI_GET_NEXT,
                              drv_name);
  .
  .
  .
/* Get if_ctlr for a specific controller instance */
.
.
error = escsi_node_to_addr(myisc->card_node,&escsi_addr, &escsi_type);
VASSERT(error == ESUCCESS);
ifctlr_reg = escsi_ifctlr_get(&escsi_addr, ESCSI_GET_THIS);
.
.
SEE ALSO

escsi_iftgt_get(9F), escsi_iflpt_get(9F), escsi_iobj_get(9F)
NAME

escsi_ifctlr_reg_t - SCSI HBA controller registration data structure

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

The escsi_ifctlr_reg_t data structure contains pointers to important data structures necessary to register a SCSI HBA controller with SCSI Services. It is passed as parameter to the escsi_ctlr_reg function. The interface driver must allocate and initialize this data structure prior to registering a SCSI HBA controller with SCSI Services.

STRUCTURE MEMBERS

typedef struct {
    uint64_t                   version;
    struct isc_table_type     *isc;
    escsi_ifctlr_attr_t       *attr;
    escsi_ifctlr_stat_t       *stat;
    escsi_ifsw_t              *ifsw;
    void                      *if_ctlr;
    escsi_trc_buf_t           trc_info;
    escsi_addr_t              addr;
} escsi_ifctlr_reg_t;

version    Specifies the API version.
isc         Specifies the ISC pointer for invoking WSIO DMA services from within the stack.
attr        Specifies a pointer to the common attributes.
stat        Specifies a pointer to the statistics.
ifsw        Specifies a pointer to the ifsw structure.
if_ctlr     Pointer to the interface driver private controller object.
trc_info    Specifies a trace buffer.
addr        Specifies a controller address.

CONSTRAINTS

None

SEE ALSO

escsi_addr_t(9S), escsi_ctlr_reg(9F), escsi_ifctlr_attr_t(9S), escsi_ifctlr_stat_t(9S), escsi_ifsw_t(9S), escsi_trc_buf_t(9S)
NAME
escsi_ifctlr_stat_t - Interface controller statistics structure

SYNOPSIS
#include <io/escsi_services.h>

DESCRIPTION
This structure contains the SCSI controller statistics that the interface driver must maintain off the escsi_ifctlr_reg_t object. These statistics help diagnose interface driver issues.

STRUCTURE MEMBERS
typedef struct {
    uint64_t    io_count;
    uint64_t    read_byte_count;
    uint64_t    write_byte_count;
    clock_t     last_reset_time;
    uint16_t    num_tgt_ports;
    uint16_t    num_bus_reset;
    uint16_t    num_offline;
    uint16_t    num_online;
} escsi_ifctlr_stat_t;

io_count        Specifies the number of outstanding I/Os.

NOTE: The interface driver must increment this counter each time it receives a new I/O request from SCSI Services and decrement it each time it sends an I/O completion or failure back to SCSI Services.

read_byte_count Specifies the number of bytes read.
write_byte_count Specifies the number of bytes written.
last_reset_time  Specifies the time when the bus last was reset.
num_tgt_ports    Specifies the number of target ports connected to this controller.
num_bus_reset    Specifies the number of bus resets that have been attempted on this controller.
num_offline      Specifies the number of times this controller has gone offline.
num_online       Specifies the number of times this controller has gone online.

CONSTRAINTS
None

SEE ALSO
escsi_ifctlr_reg_t(9S)
**NAME**

escsi_ifctlr_type_t - Controller type enumerations

**SYNOPSIS**

```c
#include <io/escsi_services.h>
```

**DESCRIPTION**

These enumerations are set by interface driver. These flags define whether the corresponds to a physical interface (for example, Fibre Channel HBA) or to a logical interface (for example, iSCSI driver).

**STRUCTURE MEMBERS**

```c
typedef enum {
    ESCSI_PHYSICAL      = 0x01,
    ESCSI_LOGICAL       = 0x02
} escsi_ifctlr_type_t;
```

- **ESCSI_PHYSICAL**
  Indicates that the HBA Controller has actual physical hardware associated with it.

- **ESCSI_LOGICAL**
  Indicates a virtual hardware component. For example, an iSCSI software driver running over a host TCP/IP stack.

**CONSTRAINTS**

None

**SEE ALSO**

* escsi_ifctlr_reg_t(9S)
NAME

escsi_iflpt_get - Get interface lun path handle for specified lun path

SYNOPSIS

void *escsi_iflpt_get(
    escsi_addr_t *addr,
    escsi_fetch_hint_t hint
);

PARAMETERS

addr  Pointer to the address structure containing the identification of a target path or a lunpath in the form of controller/target/lunpath instance numbers (c-t-l).

hint  Specifies the fetch hint.

DESCRIPTION

This function can be used by an interface driver to get its private handle for a lunpath. The interface driver registers the private handle with SCSI Services by setting to a non-NULL value in the if_lpt field of the escsi_ifspoc_ws_t data structure during completion of the lunpath open (see escsi_ifsw_t(9S), escsi_ifspoc_ws_t(9S), and xxx_if_lpt_open(9F)).

The function returns the interface private handle for a lunpath based on the address information provided in the addr parameter and the fetch hint.

The fetch hints allowed are:

<table>
<thead>
<tr>
<th>ESCSI_GET_THIS</th>
<th>Returns the private handle for the lunpath specified by addr-&gt;c, addr-&gt;t, or addr-&gt;l. It returns NULL if the lunpath requested has not been registered or the interface driver has not registered a private handle for the lunpath.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESCSI_GET_FIRST</td>
<td>Returns the private handle for the first lunpath under the target path specified by addr-&gt;c and addr-&gt;t. It returns NULL if no lunpath has been registered for the target path or the interface has not registered a private handle for the first lunpath under the target.</td>
</tr>
<tr>
<td>ESCSI_GET_NEXT</td>
<td>Returns the private handle for the lunpath next to the lunpath specified by addr-&gt;c, addr-&gt;t, and addr-&gt;l. It returns NULL if no more lunpaths are available on the target path or no interface driver private handle has been registered for the next lunpath.</td>
</tr>
</tbody>
</table>

The following table shows the valid parameter combinations:

<table>
<thead>
<tr>
<th>type</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESCSI_GET_THIS</td>
<td>addr-&gt;c, addr-&gt;t, addr-&gt;l</td>
</tr>
<tr>
<td>ESCSI_GET_FIRST</td>
<td>addr-&gt;c, addr-&gt;t</td>
</tr>
<tr>
<td>ESCSI_GET_NEXT</td>
<td>addr-&gt;c, addr-&gt;t, addr-&gt;l of the previously fetched element</td>
</tr>
</tbody>
</table>

The ESCSI_GET_FIRST and ESCSI_GET_NEXT hints refer to the order of the elements in the hash table, not the order of the element sorted by addr addresses. When passing the ESCSI_GET_NEXT hint, the caller must pass the addr of a valid lunpath element previously fetched.
RETURN VALUE

If successful, the routine returns an interface driver private handle. On error, the routine returns NULL.

CONSTRAINTS

This function can be called in ICS context or blocking context.
The interface driver cannot hold spinlocks while calling this function.

EXAMPLE

escsi_addr_t addr;
/* Get if_lpts for all lun path instances in a loop */

error = escsi_node_to_addr(myisc->card_node, &escsi_addr, &escsi_type);
VASSERT(error == ESUCCESS);
escsi_addr.t = mytgt;
if_lpt = escsi_iflpt_get(&escsi_addr, ESCSI_GET_FIRST);
while (if_lpt != NULL) {
    ...
    ...
    if_lpt = escsi_iflpt_get (&escsi_addr, ESCSI_GET_NEXT);
}

/* Get if_lpt for a specific lun path instance */

error = escsi_node_to_addr(myisc->card_node, &escsi_addr, &escsi_type);
VASSERT(error == ESUCCESS);
escsi_addr.t = mytgt;
escsi_addr.l = mylpt;
if_lpt = escsi_iflpt_get(&escsi_addr, ESCSI_GET_THIS);

SEE ALSO

escsi_addr_t(9S), escsi_ifctlr_get(9F), escsi_iftgt_get(9F), escsi_iobj_get(9F)
NAME

escsi_ifspoc_ws_t -- SCSI path open/close workspace structure.

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

Data structure passed to interface driver entry points if_lpt_open and if_lpt_close.

STRUCTURE MEMBERS

typedef struct {
    escsi_list_t       list;
    uint64_t           version;
    escsi_addr_t       addr;
    escsi_cbfn_t       cbfn
    void *             *if_lpt;
    int                errno;
    int                oflags;
    intptr_t           rsvd[5];
} escsi_ifspoc_ws_t;

list     Specifies the linked list pointers for queuing this request because the interface driver
can receive multiple concurrent lunpath open and close requests.
version  Specifies the API version.
addr     Specifies the c-t-l address of lunpath.
cbfn     Specifies the SCSI Services callback function provided to if_lpt_open and
         if_lpt_close.
if_lpt   Specifies the interface driver-specific per lunpath handle returned by the interface
driver on successful completion of a if_lpt_open. Stored in the lpt structure and
         passed back in escb_t.
errno    Specifies the status of the asynchronous function call to be returned to the caller.
oflags   Specifies open flags.
rsvd     Reserved.

CONSTRAINTS

None

SEE ALSO

escsi_iflpt_get(9F), escsi_ifsw_t(9S), xxx_if_lpt_close(9F), xxx_if_lpt_open(9F)
escsi_ifsw_t(9S)

NAME
escsi_ifsw_t - Interface driver switch table

SYNOPSIS
#include <io/escsi_services.h>

DESCRIPTION
The escsi_ifsw_t structure defines the interface driver switch table. The switch table contains
the interface driver service entry points and data structures that SCSI Services uses. The interface
driver allocates and initializes this data structure in its interface initialization or attach routine
prior to registering the SCSI HBA controller with SCSI Services. A pointer to this data structure
is specified in the escsi_ifctlr_reg_t data structure passed as parameter to the
escsi_ctlr_reg function.

STRUCTURE MEMBERS
typedef struct {
    escsi_if_flags_t  if_flags;
    escsi_task_t      if_tm_dsbl;
    uint32_t          if_max_tags;
    uint32_t          if_tag_size;
    uint32_t          if_scb_size;
    uint32_t          if_scb_bzero_size;
    uint32_t          if_max_io_size;
    uint32_t          if_timeout_msecs;
    uint32_t          if_max_conc_ios;
    void                     (*if_lpt_open)__((escsi_ifspoc_ws_t *if_spoc));
    void                     (*if_lpt_close)__((escsi_ifspoc_ws_t *if_spoc));
    void                     (*if_start)__((escsi_iobj_t *iobj));
    void                     (*if_task_mgmt)__((escsi_tm_arg_t *tm_arg));
    int              (*if_ioctl)__((dev_t dev, escsi_addr_t *addr, int cmd,
                                   caddr_t data, int flags));
    void                     (*if_ctlr_probe)__((void *if_ctlr));
    void                     (*if_leg_hwp_build)__((escsi_ifctlr_reg_t *ifctlr_reg,
                                                  escsi_lunid_t *lunid,.
                                                  escsi_lunid_t *lunid_list));
    intptr_t                rsvd[5];
} escsi_ifsw_t;

if_flags
    Specifies interface driver hints for the SCSI Services. This includes
    information such as which level of tagging is supported. See
    escsi_if_flags(9S) for more information.

if_tm_dsbl
    Specifies the exclusive OR of task management operation codes not
    supported by the interface driver and defined in escsi_task_t.
    However, support for ABORT_TASK_SET is mandatory. Do not set
    this flag in this field.

if_max_tags
    Specifies the maximum number of concurrent tags that are allowed
    to be generated. The value 0 means no limit if the interface driver has
    set a tagging flag in if_flags (see escsi_if_flags(9S)).
    If the interface driver has not set any tagging flag in the if_flags
    field, 0 means the interface driver does not support tagging.
    SCSI Services applies a maximum limit of ESCSI_MAX_TAGS tags.
    By default, ESCSI_MAX_TAGS 1024.

if_tag_size
    Specifies the tag size (number of bits used) supported by the interface
    driver. The value 0 means the number of bits do not matter.

if_scb_size
    Specifies the size of the interface driver-specific structure to be
    attached to the escsi_scb_t structure.
### if_scb_bzero_size
Specifies the number of bytes from the start of `scb->if_scb` that SCSI Services must initialize to zero. This value must be less than or equal to `if_scb_size`.

### if_max_io_size
Specifies the maximum size (in bytes) of an I/O the interface can support.

### if_timeout_msecs
Specifies the time (in milliseconds) required on this transport type for an I/O to reach the target. For transports with negligible latency, this field is zero.

### if_max_conc_ios
Specifies the maximum number of I/Os that can be passed by the SCSI Services per I/O object. This allows the SCSI Services to flow control the number of I/Os that can be sent to the select queue. If this is set to zero, no flow control is applied by SCSI Services.

### if_lpt_open
Specifies the interface driver lunpath open entry point. SCSI Services uses this interface to open a lunpath associated with the interface driver. This function is asynchronous and requires a callback into SCSI Services when the operation completes.

SCSI Services calls the lunpath open interface for every lunpath under the target path. On the first open, the interface driver needs to establish I-T nexus with the target. For subsequent opens, it might be sufficient to increment the internal reference count.

For every lunpath, the interface driver can allocate an interface specific lunpath structure during the lunpath open. SCSI Services passes as a parameter the `escsi_ifspoc_ws_t` structure, which contains the `escsi_addr_t` of the lunpath to be opened. The interface driver can pass this value to the `escsi_ifctlr_get` and `escsi_iftgt_get` SCSI Services interfaces to get interface specific data structures associated with this lunpath. Interface drivers using SCSI-2 LUN id formats receive a fake SCSI-3 64-bit LUN id in `escsi_ifspoc_ws_t->addr.l`. They must extract the SCSI-2 LUN id byte from the second most significant byte of the SCSI-3 LUN id.

SCSI Services also provides a completion callback function in `escsi_ifspoc_ws_t`. This callback function must be invoked by interface driver after the lunpath open operation is complete. The parameters to this callback function are `escsi_ifspoc_ws_t->cbfn.cb_arg` and the `errno` value indicating status of the lunpath open operation.

As part of the `if_lpt_open` function, the interface driver can create an interface specific lunpath data structure and pass that back to SCSI Services in the `if_lpt` field in the `escsi_ifspoc_ws_t` structure. This `if_lpt` value is cached by SCSI Services and passed down to the interface driver for every I/O to that lunpath. The format of the `if_lpt` structure is private to the interface driver and it can be used to store any interface specific information related to that LUN. When an I/O is sent to the lunpath, SCSI Services passes back the `if_lpt` pointer in the `scb` structure. The interface driver can use this to access other interface-specific structures in the `if_start` path.

Interface drivers do not have to use an `if_lpt` structure. If it is not used, the interface driver can return `if_lpt` as NULL. Interface drivers must return as soon as possible from asynchronous entry points such as `if_lpt_open`, `if_lpt_close`, and `if_task_mgmt` to provide maximum parallelism in the SCSI stack. This implies that interface drivers must not block in these entry points.
The **if_lpt_open** entry point is mandatory.

### if_lpt_close

Specifies the interface driver lunpath close entry point. SCSI Services uses this interface to close a lunpath associated with the interface driver. This function is asynchronous and requires a callback into SCSI Services when the operation completes. During the last lunpath close on a target path, the interface driver must close the I-T nexus with the target.

The input parameter to the lunpath close function is the `escsi_ifspoc_ws_t` structure, which contains the `escsi_addr_t` of the lunpath to be closed. The interface driver can pass this value to the `escsi_ifctlr_get`, `escsi_iftgt_get`, and `escsi_iflpt_get` SCSI Services interfaces to get interface specific data structures associated with this lunpath.

SCSI Services also provides a completion callback function in `escsi_ifspoc_ws_t`. The interface driver must invoke this callback function after the lunpath close operation is complete. The parameters to this callback function are `escsi_ifspoc_ws_t->cbfn.cb_arg` and the `errno` value indicating status of the lunpath close operation.

Interface drivers using SCSI-2 LUN id formats receive a fake SCSI-3 64-bit LUN id in `escsi_ifspoc_ws_t->addr.l`. They must extract the SCSI-2 LUN id byte from the second most significant byte of the SCSI-3 LUN id.

The **if_lpt_close** entry point is mandatory.

### if_start

Specifies the interface driver I/O start entry point. SCSI Services calls this entry point to issue I/Os on the controller or target path. I/Os are issued to the interface driver via the `select_q`, which is a queue of `scb` structures (`escsi_scb_t`). Multiple I/Os can be queued to the `select_q` before the interface driver start routine is called. This allows multiple I/Os to be issued to the interface driver with a single call to `if_start`. The input parameter for the `if_start` function is the I/O object (`escsi_iobj_t`) structure. This I/O object structure contains the `select_q` and a pointer to the interface-specific portion of the controller structure. It also contains a spinlock, which is used to synchronize access to the `select_q`.

SCSI Services provides the `ESCSI_IOBJ_LOCK` and `ESCSI_IOBJ_UNLOCK` macros to lock and unlock the I/O object while accessing the `select_q`. Interface drivers must hold the I/O lock while removing I/Os from the `select_q`.

When a new I/O is added to the `select_q`, SCSI Services ensures that the `if_start` routine is called. If an interface driver leaves I/Os behind on the `select_q` after completing `if_start` (for reasons like no internal resources available), the interface driver must check the `select_q` again in its I/O completion path to start the pending I/Os. SCSI Services specifies the timeout for each I/O in `scb->max_msecs` field. This timeout period refers to the time period allocated for the I/O after it is issued to the HBA. However, if an I/O has the `B_PFTIMEOUT` flag set in its `buf` structure, the time period is measured from the time the driver received the request in the `select_q`. This implies that the timeout period starts even while the I/O waits for resources to be available in the interface driver. If `scb->max_msecs` is specified as 0, the interface driver is not required to time that I/O.
The *if_start* entry point is mandatory.

### if_task_mgmt

Specifies the interface driver task management entry point. SCSI Services uses this entry point to issue task management functions to the LUNs or targets accessed through the SCSI HBA controller. A task management function is issued when upper layers require a task or some group of tasks to be aborted or terminated. See `escsi_task_mgmt_t` for the list of task management functions that can be issued.

The task management interface is asynchronous, and SCSI Services provides a callback function that the interface driver must call after the task management operation is over. Any I/Os failed back to SCSI Services as a result of task management operation must be returned with the SCT1_LIO_ABORTED error code. When a task management function is received, the interface driver must examine its internal queue of pending and active I/Os and return as an error those I/Os affected by the scope of the task management operation. For each I/O, the interface driver must examine whether the I/O is impacted by comparing lun-id or *if_lpt* and *if_tp* fields in the *scb*. The driver must error back the *scb*’s destined for the lunpath being impacted (`ABORT_TASK_SET`, `CLEAR_TASK_SET`, or `LUN_RESET`) or the tgtpath being impacted (`WARM_TARGET_RESET` or `COLD_TARGET_RESET`).

When SCSI Services issues task management commands, it expects interface drivers to do the following:

1. Scan the *select_q* and error back all I/Os queued for the task set or target path, depending on the scope of task management. For `ABORT_TASK_SET`, `CLEAR_TASK_SET`, or `LUN_RESET`, return I/Os on the lunpath as an error. For `WARM_TARGET_RESET` and `COLD_TARGET_RESET`, return I/Os on the target path as an error. For `BUS_RESET`, return all I/Os on the controller as an error.

2. Error back all pending (not yet active) I/Os queued internally within the driver for the task set or target path depending on the scope of task management.

3. Issue the task management command to the device.

4. If task management response from device indicates success, the interface driver must walk its internal queue of active I/Os and abort all I/Os issued for the task set or target path depending on the scope of task management.

5. If task management command fails, the interface driver must terminate I-T nexus with the device and error back all active I/Os on that target path, after waiting for a protocol-specified time period or after establishing I-T nexus again.

6. After all active I/Os on the appropriate scope have been errored back, the interface driver must invoke the task management callback function. The result of task management operation is passed back as the *errno* parameter for the callback function.

If the task management command fails, the interface driver must escalate and take a higher level error recovery as appropriate to the transport, in order to ensure that the task set (or a higher scope) of I/Os are explicitly aborted at the target end. The interface driver must initiate an I-T nexus logout or a bus reset (in case of sessionless SCSI...
transports). If a bus reset is performed, the task management is assumed to have completed explicitly.

In case of session oriented SCSI transports, the failure of a task management command requires the interface driver to log out the I-T nexus and abort the I/Os in the task set in an explicit manner. The driver must ensure that I/Os are explicitly aborted at the target end before erroring back the I/Os. This involves issuing a transport-specific logout (for example, LOGO/PLOGI for FibreChannel) and erroring back the I/Os only upon completion of the LOGO or PLOGI. If LOGO or PLOGI is unsuccessful, the interface driver must hold on to the I/Os until a transport-defined safe time period has elapsed since the last response was received from the target. Then it errors the I/Os back. This safe time period (for example, 2*RA_TOV for FibreChannel) is required to guarantee that the data is not present as ghost I/O in the interconnect fabric/loop/bus.

Interface drivers must guarantee that the task management callback function is called only after all pending and active I/Os for the specified scope have been errored back to SCSI Services. If the task management command has failed and interface driver needs to delay the I/Os by a protocol-specified safe time period, the callback function must be called only after all such I/Os waiting for this delay have been errored back.

The `if_task_mgmt` entry point is mandatory.

### `if_ioctl`

Specifies the interface driver IOCTL entry point. This entry point is required only if the interface driver specific IOCTLs are supported on the LUN device file. The `if_ioctl` entry point can be used for the following purposes:

- **To manage legacy ioctl**s issued to the ctrl through the legacy lunpath DSFs. The SCSI Services select the interface driver instance corresponding to the legacy lunpath DSF and forward these ioctl to the selected interface driver instance through the `if_ioctl` entry point. The following ioctl to supported by the SCSI Services:

  - `SIOC_GET_PORT_WWN` — Get the target port WWN of the target path corresponding to a given legacy lunpath DSF. This ioctl is mandatory for FibreChannel drivers.

  - `SIOC_GET_WWN` — Get the target node WWN of the target path corresponding to a given legacy lunpath DSF. This ioctl is mandatory for FibreChannel drivers.

  - `SIOC_RESET_BUS` — Issue a task management BUS RESET to the interface driver.

- **To support management operations that the SCSI Services need to forward to the interface driver.** The following ioctl is supported by the SCSI Services:

  - `SIOC_DEL_IFLPT` — Invoked by the SCSI Services in the context of a lunpath deletion if the interface driver had previously registered a private object with the lunpath.
**if_ctlr_probe**

Specifies the interface driver SCSI HBA controller probe entry point. As part of the probe, the interface driver is expected to discover targets accessible through the SCSI HBA controller and register their target paths with SCSI Services. This entry point is asynchronous and SCSI Services provides a `escsi_if_ctlr_prb_cbfn` callback function that the interface driver must call on completion of the controller probe (after the registration of the last target path). The parameters to this callback function are a pointer to SCSI HBA controller registration data structure (`escsi_ifctlr_reg_t`) and the `errno` value indicating the probe completion status. The *if_ctlr_probe* entry point is mandatory.

**if_leg_hwp_build**

Specifies the interface driver legacy hardware path build entry point. SCSI Services calls this entry point to build legacy format hardware paths for each of the LUNs specified in the list of LUN ids provided. The interface driver uses this LUN id list to build legacy hardware paths for each of the LUNs and returns back to SCSI Services. The interface driver cannot sleep in the context of the *if_leg_hwp_build* entry point. The *if_leg_hwp_build* entry point is mandatory.

**rsvd**

Reserved.

**CONSTRAINTS**

None

**SEE ALSO**

`escsi_ifctlr_reg_t(9S), escsi_if_flags_t(9S), escsi_task_t(9S)`
**NAME**

escsi_iftgt_attr_t - Target path attributes

**SYNOPSIS**

```c
#include <io/escsi_services.h>
```

**DESCRIPTION**

This structure contains the target path attributes provided by the interface driver before invoking the `escsi_tgt_reg` KPI. The `escsi_iftgt_attr_t` structure is passed in by reference to the `escsi_tgt_reg` routine. Interface drivers must allocate and initialize the structure and pass it in during target registration. The `escsi_iftgt_attr_t` structure is owned by interface drivers, but can be referred to by SCSI Services. The interface driver can update this structure on-the-fly to keep the attributes up-to-date.

**STRUCTURE MEMBERS**

```c
typedef struct {
    escsi_name_t          port_name;
    uint64_t              port_id;
    escsi_name_t          node_name;
    char                  *protocol_rev;
    uint32_t              iobj_id;
} escsi_iftgt_attr_t;
```

- **port_name** Specifies the target port name. Cannot be modified by the interface driver.
- **port_id** Specifies the target port ID (for example, FC nport ID pSCSI target number). Cannot be modified by the interface driver. However, the interface driver can update this field with AEN. The interface driver must call `escsi_if_aen` with the `ESCSI_PORT_ID_CHANGE` event.
- **node_name** Specifies the target node name. Cannot be modified by the interface driver.
- **protocol_rev** Specifies the protocol revision number. Can be modified by the interface driver during an online replacement (OLR).
- **iobj_id** Specifies the I/O object ID associated with the target path. Cannot be modified by the interface driver. After it is set, the interface driver must not dynamically change the association between an I/O object and a target path object.

**CONSTRAINTS**

None

**SEE ALSO**

`escsi_iftgt_reg_t(9S), escsi_name_t(9S), escsi_tgt_reg(9F)`
NAME

escsi_iftgt_get - Get interface target path handle for specified target path

SYNOPSIS

int escsi_iftgt_get(
    escsi_addr_t *addr,
    uint64_t portid,
    escsi_name_t *name,
    escsi_fetch_hint_t hint,
    escsi_get_flag_t flag,
    escsi_iftgt_reg_t **iftgt_reg
);

PARAMETERS

addr Pointer to an eSCSI address structure.
portid Target port ID.
name Target port name.
hint Fetch hint.
flag Target address element type. ESCSI_ADDR_LOOKUP, ESCSI_NAME_LOOKUP, or
      ESCSI_PORTID_LOOKUP.
iftgt_reg Pointer to the requested escsi_iftgt_reg_t structure.

DESCRIPTION

This function can be used by interface drivers to get the interface-specific registration handle for
the target path structure. The caller can specify the required target path by passing either an
escsi_addr_t instance address or target port id or target port name. This service returns an
iftgt_reg handle based on the combination of a fetch hint, a flag, and some target port id or name.
The valid combinations of input parameters are:

- ESCSI_GET_THIS, ESCSI_ADDR_LOOKUP, addr->c, addr->t
  Returns the ifgt_reg handle for the target path corresponding to instance addr->t under
  the controller instance addr->c.

- ESCSI_GET_THIS, ESCSI_NAME_LOOKUP, addr->c, name
  Returns the ifgt_reg handle of the target path which is attached to the controller addr->c
  instance, and whose name matches the name parameter.

- ESCSI_GET_THIS, ESCSI_PORTID_LOOKUP, addr->c, portid
  Returns the ifgt_reg handle corresponding to the target path which is attached to the
  controller of addr->c instance, and whose port ID matches the portid parameter.

- ESCSI_GET_FIRST, addr->c
  Returns the ifgt_reg handle of the first target path in the target path hash table of the specified
  addr->c controller instance

- ESCSI_GET_NEXT, addr->c, addr->t
  — return the ifgt_reg handle corresponding to the target path following the target path
  component of address addr->t in the target path hash table under controller instance
  addr->c.

The ESCSI_GET_FIRST and ESCSI_GET_NEXT hints refer to the order of the elements in the
SCSI Services hash table, not the order of the element sorted by addr address values. When passing
the `ESCSI_GET_NEXT` hint, the caller must pass the `addr` of a valid target path element previously fetched.

**RETURN VALUE**

**ESUCCESS** Successful.

**EINVAL** Invalid parameters. For example:
- Null address pointer
- Invalid controller instance number
- Invalid previous (c-t) nexus in the context of a `ESCSI_GET_NEXT` lookup
- NULL name with the `ESCSI_NAME_LOOKUP` flag
- Invalid type or flag parameters

**ENODEV** The `escsi_iftgt_reg_t` pointer is not found.

**CONSTRAINTS**

This function can be called in ICS context or blocking context.
The interface driver cannot hold spinlocks while calling this function.

**EXAMPLE**

```c
escsi_addr_t addr;

/* Test iftgt_reg in a loop for all tgtpath instances */
error = escsi_node_to_addr(myisc->card_node, &escsi_addr, &escsi_type);
VASSERT(error == ESUCCESS);
ret = escsi_iftgt_get(&escsi_addr, 0, NULL, ESCSI_GET_FIRST,
                      ESCSI_ADDR_LOOKUP, &iftgt_reg);
while (ret == ESUCCESS) {
    
    ret = escsi_iftgt_get (iftgt_reg->addr, 0, NULL, ESCSI_GET_NEXT,
                           ESCSI_ADDR_LOOKUP, &iftgt_reg);
}

/* Get iftgt_reg for a specific tgtpath instance */

error = escsi_node_to_addr(myisc->card_node, &escsi_addr, &escsi_type);
VASSERT(error == ESUCCESS);
escsi_addr.t = mytgt;
ret = escsi_iftgt_get(addr, 0, NULL, ESCSI_GET_THIS
                      ESCSI_ADDRESS_LOOKUP, &iftgt_reg);
```

**SEE ALSO**

`escsi_ifctlr_get(9F), escsi_iflpt_get(9F), escsi_iobj_get(9F)`
NAME

escsi_iftgt_reg_t - Interface target registration structure

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

The escsi_iftgt_reg_t structure is a parameter to the escsi_tgt_reg routine. Interface
drivers must allocate and initialize the structure and pass it in during target registration. This
structure contains links to other structures containing target attributes (escsi_iftgt_attr_t)
and target related statistics (escsi_iftgt_stat_t), which the interface driver must also
allocate. The escsi_iftgt_attr_t and escsi_iftgt_stat_t structures are owned by
interface drivers, but can be referred to by SCSI Services. The interface driver can update these
structures on-the-fly to keep the attributes and statistics up-to-date.

STRUCTURE MEMBERS

typedef struct {
    void                    *node;
    int (* wsio_cb_func)(void *handle, void *arg, io_events_t event,
                          uintptr_t event_info);
    io_events_t             events_mask;
    void                    *arg;
    escsi_hwp_t             hwp;
    escsi_thcb_t            thcb;
    uint64_t                version;
    escsi_iftgt_attr_t      *attr;
    escsi_iftgt_stat_t      *stat;
    escsi_cbfunc_t          if_tgt_reg_cbfn;
    void                    *if_tgt;
    escsi_addr_t            addr;
} escsi_iftgt_reg_t;

node Specifies the target path I/O node.

wsio_cb_func Specifies the callback routine for the target path. This is mandatory.

events_mask Specifies the events supported by the interface driver. I/O node callback
            routine events. The CB_DESTROY operation code must be supported.

arg Specifies the arguments for the callback routine. This is mandatory.

hwp Specifies the extended hardware path elements for the target path. This
        is mandatory.

thcb Reserved for SCSI Services.

version Specifies the API version.

attr Specifies the target attributes. This is mandatory.

stat Specifies the target statistics. This is mandatory.

if_tgt_reg_cbfn Specifies the interface driver target register callback routine. This routine
takes the following arguments: pointer to escsi_tgt_reg_t and an
errno.

if_tgt Pointer to the interface driver private target path object.

addr When called, specifies the controller instance number; this is mandatory.
        On return, contains the target path instance number.
CONTRAINTS

None

SEE ALSO

escsi_addr_t(9S), escsi_cbfunc_t(9S), escsi_hwp_t(9S), escsi_ifigt_attr_t(9S), escsi_ifigt_stat_t(9S), escsi_ifigt_reg_t(9S), escsi_tgt_reg(9F), escsi_thcb_t(9S)
escsi_iftgt_stat_t(9S)

NAME

escsi_iftgt_stat_t - Interface driver's target path statistics

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

The escsi_iftgt_stat_t structure is passed in by reference to the escsi_tgt_reg routine. Interface drivers must allocate and initialize the structure and pass it in during target registration. The escsi_iftgt_stat_t structure is owned by interface drivers, but can be referred to by SCSI Services. The interface driver can update this structure on-the-fly to keep the statistics up-to-date.

STRUCTURE MEMBERS

typedef struct {
    uint16_t              offline_count;
    uint16_t              num_tgt_reset;
    struct timeval        last_reset_time;
    uint64_t              read_byte_count;
    uint64_t              write_byte_count;
    uint64_t              io_count;
} escsi_iftgt_stat_t;

offline_count         Specifies the offline event count.
num_tgt_reset         Specifies the number of warm or cold BDR.
last_reset_time       Specifies the time of the last warm or cold BDR.
read_byte_count       Specifies the number of bytes read.
write_byte_count      Specifies the number of bytes written.
io_count              Specifies the number of outstanding I/Os on this target path.

IMPORTANT: The interface driver must increment this counter when a new I/O request is received on this target path, and must decrement it when an I/O is completed or when failure is returned to SCSI Services.

CONSTRAINTS

None

SEE ALSO

escsi_iftgt_reg_t(9S), escsi_tgt_reg(9F)
NAME

escsi_info_t - SCSI object general information

SYNOPSIS

#include <io/escsi_stack.h>

STRUCTURE MEMBERS

type  Specifies the type of the SCSI object. Valid values are ESCSI_GLOBAL, ESCSI_LUN, and ESCSI_LPT. This field identifies a SCSI object from its information field.

DESCRIPTION

The escsi_info_t structure holds general SCSI object information. It is located at the beginning of most SCSI objects (see escsi_lun_t(9S), escsi_global_t(9S), and escsi_lpt_t(9S)). Class drivers can access the previous fields.

SEE ALSO

escsi_global_t(9S), escsi_lpt_t(9S), escsi_lun_t(9S)
ESCSI_INIT_ELEM(9G)

NAME
ESCSI_INIT_ELEM - Macro to initialize linked list element pointers.

SYNOPSIS
ESCSI_INIT_ELEM(  
    escsi_list_t *__elem  
);  

PARAMETERS
__elem Device number.

DESCRIPTION
The ESCSI_INIT_ELEM macro initializes the previous and next pointers of a list element to NULL. You can use this macro to set up an element before enqueueing it to the linked list.

RETURN VALUE
None

CONSTRAINTS
None

EXAMPLE
escsi_list_t list_elem;
ESCSI_INIT_ELEM(&list_elem);

SEE ALSO
ESCSI_INIT_HDR(9G)

NAME
ESCSI_INIT_HDR - Macro to initialize linked list header to point to itself.

SYNOPSIS
ESCSI_INIT_HDR(
    escsi_list_t *__hdr
);

PARAMETERS
__hdr Pointer to list header.

DESCRIPTION
The ESCSI_INIT_HDR macro initializes the previous and next pointers of a list header to point to itself.

RETURN VALUE
None

CONSTRAINTS
None

EXAMPLE
escsi_list_t queue_header;
ESCSI_INIT_HDR(&queue_header);

SEE ALSO
NAME
escsi_init_sema - Initialize a SCSI subsystem semaphore.

SYNOPSIS
void escsi_init_sema(
    escsi_bsema_t *sema
);

PARAMETERS
sema Pointer to an eSCSI beta semaphore.

DESCRIPTION
This routine initializes a semaphore.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE

SEE ALSO
escsi_owns_sema(9F), escsi_psema(9F), escsi_vsema(9F)
NAME

escsi_iobj_cpu_t - Interface object CPU registration structure

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

This structure contains the parameters for the ESCSI_IOBJ_CPU_REG operation for escsi_if_aen. At interface driver initialization time, after the interrupt object has been allocated, the interface driver queries the I/O Infrastructure for the assigned CPU id. After this, it notifies SCSI Services to bind the I/O object to that CPU by sending ESCSI_IOBJ_CPU_REG event through escsi_if_aen. The details of I/O object and CPU id are filled in this structure and a pointer to this is passed in the evt_data field of escsi_aen_t. When interrupt migration happens, interface drivers must perform this I/O object binding again with the new CPU id by calling escsi_if_aen again.

STRUCTURE MEMBERS

typedef struct {
    uint32_t            iobj_id;
    spu_t               cpu_id;
} escsi_iobj_cpu_t;

iobj_id    Specifies the I/O object ID.
cpu_id     Specifies the CPU number.

CONSTRAINTS

None

SEE ALSO

escsi_aen_t(9S), escsi_if_aen(9F)
NAME
escsi_iobj_get - Get pointer to the escsi_iobj_t data structure

SYNOPSIS
<sys/escsi_services.h>

escsi_iobj_t *escsi_iobj_get(
    escsi_ifctlr_reg_t *ifctlr,
    uint32_t iobj_id
);

PARAMETERS
ifctlr Pointer to the SCSI HBA controller registration data structure
(escsi_ifctlr_reg_t).
iobj_id Index of the I/O object. This index must be within the number of I/O objects allocated
during the SCSI HBA controller registration (ifctlr->attr->num_iobj).

DESCRIPTION
This function can be called by an interface driver to get the pointer to the
escsi_iobj_t data structure corresponding to an I/O object whose index (iobj_id) is passed in as a parameter.

RETURN VALUE
On successful completion, the routine returns a pointer to the requested escsi_iobj_t structure.
Returns a NULL pointer if iobj_id is out of range or the controller is not registered with SCSI
Services.

CONSTRAINTS
This function can be called in ICS context or blocking context.
The interface driver cannot hold spinlocks while calling this function.

EXAMPLE

escsi_ifctlr_reg_t  *ifctlr;
escsi_iobj_t       *iobj;
.
.
/* Get pointer to I/O object with id 0 */
iobj = escsi_iobj_get(ifctlr, 0);

SEE ALSO
escsi_ifctlr_get(9F), escsi_iftgt_get(9F), escsi_lpt_get(9F), ESCSI_IOBJ_LOCK(9G),
ESCSI_IOBJ_UNLOCK(9G), ESCSI_OWNS_IOBJ_LOCK(9G)
NAME
ESCSI_IOBJ_LOCK - Macro to lock I/O object spinlock

SYNOPSIS
ESCSI_IOBJ_LOCK (escsi_iobj_t *__iobj)

PARAMETERS
_iobj Pointer to I/O object structure.

DESCRIPTION
The ESCSI_IOBJ_LOCK macro locks the I/O object spinlock. Interface drivers must use this macro in the if_start function to lock the I/O object prior to removing I/Os from select_q.

The lock order of I/O object lock is higher than the order of spinlocks used by interface drivers. This allows interface drivers to lock I/O object even while holding their interface driver-specific lock.

RETURN VALUE
None

CONSTRAINTS
Can be called while holding interface driver-specific spinlocks.

EXAMPLE
ESCSI_IOBJ_LOCK(iobj);

SEE ALSO
ESCSI_IOBJ_UNLOCK(9G), ESCSI_OWNS_IOBJ_LOCK(9G)
NAME

escsi_iobj_t - Shared I/O object definition

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

The shared I/O object definition encapsulates the select_q and interface driver resources related to the queue pair to which the I/O is being issued. For controllers that support multiple queue pairs, each I/O object maps to one queue pair.

STRUCTURE MEMBERS

typedef struct {
  int version;
  lock_t *lock;
  spu_t cpu_id;
  uint32_t iobj_id;
  escsi_list_t select_q;
  void *if_ctlr;
  void *if_iobj;
  intptr_t rsvd[5];
} escsi_iobj_t;

version Specifies the ABI version.
lock Specifies the synchronization lock for the I/O object.
cpu_id Specifies the ID of the CPU to which this I/O object is bound. The I/Os issued on this I/O object are forwarded to specified CPU.
iobj_id Specifies the I/O object ID assigned by the SCSI Services.
select_q Specifies the I/O queue, containing I/Os queued by SCSI Services, to be issued by the interface driver. The select_q field is a list of doubly linked scb structures.
if_ctlr Specifies the back pointer to the interface driver’s private controller structure.
if_iobj Specifies the back pointer to the interface driver’s private I/O object.
rsvd Reserved.

CONSTRAINTS

None

SEE ALSO

escsi_list_t(9S)
ESCSI_IOCTL_UNLOCK(9G)

NAME
ESCSI_IOCTL_UNLOCK - Macro to unlock I/O object spinlock

SYNOPSIS
ESCSI_IOCTL_UNLOCK (escsi_iobj_t *__iobj)

PARAMETERS
__iobj Pointer to I/O object structure.

DESCRIPTION
The ESCSI_IOCTL_UNLOCK macro unlocks the I/O object spinlock. Interface drivers must use this macro in the if_start function to lock the I/O object prior to removing I/Os from select_q.

RETURN VALUE
None

CONSTRAINTS
Can be called while holding interface driver-specific spinlocks.

EXAMPLE
ESCSI_IOCTL_LOCK(iobj);
while (ESCSI_IS_NOT_EMPTY(iobj->select_q) {
    /* Remove I/O from select_q and process it */
    .
    .
}ESCSI_IOCTL_UNLOCK(iobj);

SEE ALSO
ESCSI_IOCTL_LOCK(9G), ESCSI_OWNS_IOCTL_UNLOCK(9G)
NAME

escsi_ioctl - Processes all the common ioctl commands.

SYNOPSIS

int escsi_ioctl(
    dev_t dev,
    escsi_lpt_t *lpt,
    int cmd,
    caddr_t udata_ptr,
    int flags
);  

PARAMETERS

dev The dev_t on which the ioctl is issued.
lpt Pointer to escsi_lpt structure.
cmd Specifies the ioctl command to process.
udata_ptr Pointer to the data buffer.
flags Flags

DESCRIPTION

This routine processes the common legacy and new ioctls. If escsi_ioctl cannot process the
ioctl, the routine forwards it to the interface driver. If the interface driver fails to process the ioctl,
escsi_ioctl passes it to the class driver that owns the LUN using the
escsi_ddsw_t->dd_ioctl entry point.
This routine must be invoked by the class driver in the drv_ops_t->d_ioctl entry point.

RETURN VALUE

ESUCCESS Interface command was successfully processed.
EINVAL Interface command not supported.

CONSTRAINTS

This routine is called with a semaphore held if the caller is a pass-through driver.
This routine is called in a process context. If dev is a legacy dev_t, lpt is non-NULL and
corresponds to a legacy dev_t. If dev is a new style dev_t, lpt is NULL.

EXAMPLE

int mydriver_ioctl(dev_t dev, int cmd, caddr_t udata_ptr, int flags)
{
    int errno;

    if(ESCSI_IS_MGMT_DEV(dev)) {
        /* Handle management ioctl */
        .
        .
        .
        return errno;
    }

    return escsi_ioctl(dev);
}
SEE ALSO

escsi_ddsw_t(9S), ESCSI_IS_MGMT_DEV(9G)
NAME
escsi_iiodone - Complete an I/O request.

SYNOPSIS
void escsi_iiodone(
    struct buf *bp
);

PARAMETERS
bp  Pointer to the buf structure of the I/O request being completed.

DESCRIPTION
This routine is invoked by class drivers to notify the upper layers of an I/O completion.

RETURN VALUES
None

CONSTRAINTS
Do not hold a lock when calling this routine.

EXAMPLE
int
mydriver_dd_strategy(escsi_lun_t *lun, struct buf *bp)
{

  if(error condition) {
    ...  
    bp->b_flags |= B_ERROR;
    bp->b_error = ENXIO;
    ESCSI_LUN_UNLOCK(lun);
    escsi_iiodone(bp);
    ESCSI_LUN_LOCK(lun);
    return -1;
  }

  return 0;
}

SEE ALSO

964 SCSI Reference Pages
ESCSI_IS_EMPTY(9G)

NAME
ESCSI_IS_EMPTY - Macro to check if a linked list is empty

SYNOPSIS
ESCSI_IS_EMPTY(
    escsi_list_t *__hdr
);

PARAMETERS
__hdr Pointer to the list header.

DESCRIPTION
The ESCSI_IS_EMPTY macro check if the given list is empty.

RETURN VALUE
ESCSI_TRUE The list header is empty.
ESCSI_FALSE The list header is not empty.

EXAMPLE
escsi_list_t list_hdr;
escsi_list_t *elem;

if (ESCSI_IS_EMPTY(&list_hdr) == ESCSI_FALSE)
    elem = ESCSI_NEXT(&list_hdr);

CONSTRAINTS
None

SEE ALSO
ESCSI_NEXT(9G)
NAME
escsi_is_legacy_dev - A boolean function that determines if the devt belongs to a legacy class driver.

SYNOPSIS
escsi_bool_t escsi_is_legacy_dev(
    dev_t dev,
    int  dev_type
);

PARAMETERS
  dev       The devt of the lun.
  dev_type  D_BLK or D_CHR

DESCRIPTION
This SCSI Services public routine checks if the devt belongs to a legacy class driver.

RETURN VALUE
ESCSI_TRUE   The devt belongs to a legacy class driver.
ESCSI_FALSE  The devt does not belong to a legacy class driver.

CONSTRAINTS
None

SEE ALSO
ESCSI_IS_LPT_ONLINE(9G)

NAME
ESCSI_IS_LPT_ONLINE - Macro to check if a lunpath is online.

SYNOPSIS
ESCSI_IS_LPT_ONLINE(__lpt)

PARAMETERS
__lpt Pointer to a escsi_lpt_t structure.

DESCRIPTION
The ESCSI_IS_LPT_ONLINE macro checks whether a lunpath is ready to send an I/O request.

RETURN VALUES
ESCSI_TRUE The lunpath is online.
ESCSI_FALSE The lunpath is not ready to send any I/O request.

CONSTRAINTS
None

EXAMPLE

escsi_lpt_t *lpt;

ESCSI_LPT_LOCK(lpt);
if (ESCSI_IS_LPT_ONLINE(lpt)) {
    /* The lunpath is ready to send an I0 request. */
    .
    .
    .
}
ESCSI_LPT_UNLOCK(lpt);

SEE ALSO
NAME
ESCSI_IS_MGMT_DEV - Macro to determine if a dev_t is a SCSI subsystem management dev_t.

SYNOPSIS
ESCSI_IS_MGMT_DEV(__dev)

PARAMETERS
__dev Pointer to a dev_t structure.

DESCRIPTION
The ESCSI_IS_MGMT_DEV macro determines if a dev_t corresponds to the /dev/escsi SCSI management device file. The scsimgr can issue ioctls through the /dev/escsi SCSI subsystem management device file.

RETURN VALUES
ESCSI_TRUE The dev_t is a SCSI subsystem management device file.
ESCSI_FALSE The dev_t is not a SCSI subsystem management device file.

CONSTRAINTS
None

EXAMPLE
See the escsi_ioctl(9F) example.

SEE ALSO
escsi_ioctl(9F), SIOC_ATTR(9G)
ESCSI_IS_NOT_EMPTY(9G)

NAME
ESCSI_IS_NOT_EMPTY - Macro to check if a linked list is not empty.

SYNOPSIS
ESCSI_IS_NOT_EMPTY(
    escsi_list_t *__hdr
);

PARAMETERS
    __hdr  Pointer to the list header.

DESCRIPTION
The ESCSI_IS_NOT_EMPTY macro checks whether the given list is empty or not.

RETURN VALUE
ESCSI_TRUE      The list is empty.
ESCSI_FALSE     The list is not empty.

EXAMPLE
if (ESCSI_IS_NOT_EMPTY(&activeq_hdr) )
    elem = ESCSI_NEXT(&activeq_hdr);

CONSTRAINTS
None

SEE ALSO
NAME
escsi_is_on_q - Determine if an element is on the queue.

SYNOPSIS

```c
escsi_bool_t escsi_is_on_q(
    escsi_list_t *qhdr,
    escsi_list_t *elem
);
```

PARAMETERS

- `qhdr`  Pointer to the queue header.
- `elem`  Pointer to the element being checked.

DESCRIPTION

This function checks if the `elem` element is present on the queue whose header is `qhdr`.

RETURN VALUE

- **ESCSI_TRUE**  The element is on the queue.
- **ESCSI_FALSE**  The element is not on the queue.

EXAMPLE

```c
if (escsi_is_on_q(&activeq_hdr, scb) == ESCSI_TRUE) {
    /* Element is found on active queue */
    .
    .
    .
}
```

CONSTRAINTS

The caller must protect the queue with the appropriate synchronization mechanism.

SEE ALSO

- `ESCSI_PREV(9G)`, `ESCSI_NEXT(9G)`, `ESCSI_INIT_HDR(9G)`, `ESCSI_IS_EMPTY(9G)`,
- `ESCSI_IS_NOT_EMPTY(9G)`, `ESCSI_ENQ_AT_HEAD(9G)`, `ESCSI_ENQ_AT_TAIL(9G)`,
- `ESCSI_ENQ_BEFORE(9G)`, `ESCSI_ENQ_AFTER(9G)`, `ESCSI_DEQ(9G)`,
- `ESCSI_DEQ_FROM_HEAD(9G)`, `ESCSI_DEQ_FROM_TAIL(9G)`,
- `ESCSI_DEQ_FROM_HEAD_FAST(9G)`, `ESCSI_DEQ_FROM_TAIL_FAST(9G)`,
- `ESCSI_MOVE_LIST(9G)`, `ESCSI_PREPEND_LIST(9G)`, `ESCSI_APPEND_LIST(9G)`,
- `ESCSI_SET_PARENT(9G)`, `ESCSI_GET_PARENT(9G)`
escsi_kr_close(9F)

NAME

escsi_kr_close - Closes a SCSI KRS node entry.

SYNOPSIS

void escsi_kr_close(
    escsi_kr_key_t *key
);

PARAMETERS

key KRS node key obtained when the node was opened.

DESCRIPTION

This routine closes a SCSI KRS node entry that was initially opened with escsi_kr_open.

RETURN VALUE

None

CONSTRAINTS

The key corresponds to an SCSI KRS entry opened or created using escsi_kr_open.

EXAMPLE

See the escsi_kr_set_attr(9F) example.

SEE ALSO

escsi_kr_delete(9F), escsi_kr_get_settable_attr(9F), escsi_kr_key_t(9S), escsi_kr_lookup_attr(9F),
escsi_kr_open(9F), escsi_kr_set_attr(9F)
NAME
escsi_kr_delete -- Delete a SCSI KRS entry

SYNOPSIS
int escsi_kr_delete(
    escsi_kr_key_t *key
);

PARAMETERS
key KRS node key for the entry to be deleted. This was obtained when the node was opened.

DESCRIPTION
This routine deletes the specified eSCSI KRS entry. All attributes and their corresponding values associated with the entry are lost.

RETURN VALUE
ESUCCESS Operation is successful.
EINVAL The specified eSCSI KRS entry key is not valid.
ENOENT The specified entry does not exist.

CONSTRAINTS
The key corresponds to an SCSI KRS entry opened or created using escsi_kr_open.

EXAMPLE
See escsi_kr_set_attr(9F) example.

SEE ALSO
escsi_kr_close(9F), escsi_kr_get_settable_attr(9F), escsi_kr_key_t(9S), escsi_kr_lookup_attr(9F),
escsi_kr_open(9F), escsi_kr_set_attr(9F)
escsi_kr_flags_t(9S)

NAME

escsi_kr_flags_t - Flags used by eSCSI KRS services

SYNOPSIS

#include <io/escsi_node.h>

STRUCTURE MEMBERS

typedef enum {
    ESCSI_KR_NOFLAG         = 0x00,
    ESCSI_KR_DRVDEF         = 0x01,
    ESCSI_KR_CREATE         = 0x02,
    ESCSI_KR_PERSISTENT     = 0x04,
    ESCSI_KR_CURRENT        = 0x08,
    ESCSI_KR_SAVED          = 0x10,
    ESCSI_KR_DEFAULT        = 0x20,
    ESCSI_KR_OPEN_OBJ       = 0x40
} escsi_kr_flags_t;

ESCSI_KR_NOFLAG No flag is specified.
ESCSI_KR_DRVDEF Set a driver default attribute.
ESCSI_KR_CREATE Create a DDR entry.
ESCSI_KR_PERSISTENT Create a DDR entry persistently or set a DDR attribute persistently.
ESCSI_KR_CURRENT Get the current value of a DDR attribute.
ESCSI_KR_SAVED Get the saved value of a DDR attribute.
ESCSI_KR_DEFAULT Get the default value of an attribute or restore the default value of an attribute.
ESCSI_KR_OPEN_OBJ Open the specified object only.

DESCRIPTION

The escsi_kr_flags_t structure defines flags used by eSCSI KR services.

SEE ALSO

escsi_kr_key_t(9S), escsi_kr_open(9F)
NAME
escsi_kr_get_settable_attr - Get values of a settable attribute from KRS.

SYNOPSIS
void escsi_kr_get_settable_attr(
    escsi_kr_key_t *key,
    char *name,
    void *cur_val,
    void *def_value,
    void *sav_val,
    uint32_t *size,
    escsi_attr_flag_t *flags,
    int *status,
    kr_type_t *type
);

PARAMETERS
key Pointer to a escsi_kr_key_t structure obtained when the KRS node was created with escsi_kr_open.
name Pointer to the attribute name.
cur_val On return, a pointer to the current attribute value.
def_val On return, a pointer to the default attribute value.
sav_val On return, a pointer to the saved attribute value.
size When called, the buffer size of buffer for the values. On return, the actual size returned by KRS.
flags On return, contains the flags corresponding to the fetched attribute values.
status On return, a pointer to attribute status.
type On return, a pointer to attribute generic type returned by KRS.

DESCRIPTION
This routine returns the values of a settable attribute from KRS.

RETURN VALUES
None

CONSTRAINTS
Caller needs to allocate enough buffer space.
The type of the attribute is returned by the function. This type correspond to KR type. The caller might need to convert it to escsi_attr_type_t.
Current values are always defined, but saved and default value might not be defined. If current value cannot be found, the error is treated as if the attribute is not available. For example, *status = EIO.

EXAMPLE
See escsi_kr_set_attr(9F) example.
SEE ALSO

*escsi_kr_set_attr*(9F)
escsi_kr_key_t(9S)

NAME

escsi_kr_key_t - Opaque handle of an open SCSI KRS entry.

SYNOPSIS

#include <io/escsi_node.h>

STRUCTURE MEMBERS

typedef struct {
    kr_key_t key_obj;
    kr_key_t key_parent;
} escsi_kr_key_t;

key_obj Key to an instance-specific KRS node.
key_parent Key to the parent KRS node. This might be a corresponding DDR entry.

DESCRIPTION

The escsi_kr_key_t structure holds an opaque handle that represents the eSCSI KRS node
being opened by escsi_kr_open.

The eSCSI Kernel Registry Services uses an opaque eSCSI KR key that contains two KRS keys,
corresponding to the object specific KRS node and the parent node used to retrieve default values
for attributes from a higher-scope KRS node. The two keys are opaque to the caller. The
escsi_kr_open routine returns an opaque escsi_kr_key_t type that is used for subsequent
eSCSI KRS calls.

SEE ALSO

escsi_kr_open(9F)
NAME

escsi_kr_lookup_attr - Retrieve the current, saved, or default value of an eSCSI attribute.

SYNOPSIS

int escsi_kr_lookup_attr(
    escsi_kr_key_t *key,
    char *attr_name,
    escsi_kr_flags_t flags,
    uint32_t *buf_size,
    void *attr_value,
    kr_type_t *attr_type
);

PARAMETERS

key Pointer to escsi_key_t structure obtained when the KRS node was created with escsi_kr_open.
attr_name Name of the attribute.
flags One of the following:
    ESCSI_KR_CURRENT Retrieve the current value of the attribute.
    ESCSI_KR_SAVED Retrieve the saved value of the attribute.
    ESCSI_KR_DEFAULT Retrieve the default value of the attribute.
buf_size When called, the size of the attr_value buffer allocated. On return, indicates the actual size of the buffer or the buffer size required if error code is ENOMEM.
attr_value Attribute value.
attr_type Attribute type.

DESCRIPTION

This routine retrieves the current, saved, or default value of an eSCSI attribute for a specified eSCSI object. The lookup for the attribute value occurs in the following order: object-specific scope, KRS driver scope (/escsi/drv_name subtree), and eSCSI global scope.

If the object-specific KRS node does not exist or if the attribute is not found for the object-specific KRS node, the driver scope is used to retrieve the value of the attribute. Inheritance is applied to retrieve the value of the attribute from a higher scope when it is not found at the given scope.

For example, /escsi/esdisk/HP has an attribute named timeout with a value of 30. If you query the timeout attribute on /escsi/esdisk/HP/revX, it returns a value of 30 even if the timeout attribute is not explicitly set on the KRS entry /escsi/esdisk/HP/revX.

RETURN VALUE

ESUCCESS Attribute value is successfully retrieved and available in attr_value.
EINVAL The specified attribute was not found.
ENOMEM Buffer size is not sufficient to hold the value of the attribute. The buf_size parameter is set to the size of the buffer required to hold the attribute value.

CONSTRAINTS

Caller knows actual size and type of attribute value and buffer passed for attribute values has enough space to accommodate the value.

Current value is always defined, but saved and default value might not be defined.
If the current value cannot be found, error is treated as if the attribute is not available. For example, *
*status = EIO.

**EXAMPLE**

See the `escsi_kr_set_attr(9F)` example.

**SEE ALSO**

`escsi_kr_close(9F)`, `escsi_kr_delete(9F)`, `escsi_kr_get_settable_attr(9F)`, `escsi_kr_flags_t(9S)`, `escsi_kr_key_t(9S)`, `escsi_kr_lookup_attr(9F)`, `escsi_kr_set_attr(9F)`
NAME

escsi_kr_open - Open or create an eSCSI KRS entry.

SYNOPSIS

int escsi_kr_open(
    escsi_krid_t *krid,
    escsi_type_t type,
    escsi_kr_flags_t flags,
    escsi_kr_key_t *key
);

PARAMETERS

krid  Pointer to escsi_krid_t structure that identifies the eSCSI object instance. When the object type is ESCSI_LUN, the escsi_info_t structure passed in actually corresponds to the LUN path of the LUN.

type  SCSI object type.

flags  One of the following:

   ESCSI_KR_CREATE        Create the entry if it does not already exist.
   ESCSI_KR_PERSISTENT    The created entry persistent across system reboots.
   ESCSI_KR_OPEN_OBJ      Open object specific entry only; do not open parent KRS node.

key  On return, pointer to the KRS key pair for the instance-specific KRS node and the parent.

DESCRIPTION

The escsi_kr_open routine opens KRS nodes corresponding to the eSCSI object-specific entry and the parent entry. The routine returns the corresponding KRS keys in escsi_kr_key_t. If the object-specific entry does not exist, KR_NOKEY is returned in key->key_obj.

If the eSCSI object type specified by escsi_info_t is LUN or LUN_PATH, the parent node corresponds to the driver-specific (/escsi/drv_name subtree) KRS entry for the LUN or lunpath. If the eSCSI object type is TARGET_PATH or CONTROLLER, the parent key corresponds to the eSCSI KRS node at global scope (/escsi).

Specifying the ESCSI_KR_CREATE flag creates the eSCSI KRS entry if it does not already exist. Specifying the ESCSI_KR_PERSISTENT flag along with ESCSI_KR_CREATE creates the entry persistently.

RETURN VALUE

ESUCCESS  The create or open call was successful.

EINVAL  One of the following invalid parameters was passed:

   • Invalid name
   • Depth of the target node exceeds KR_MAX_DEPTH
   • Name component of path exceeds KR_NAME_LEN
   • Name is not a valid KRS path

ENOENT  There is no object-specific KRS entry, or the ESCSI_OPEN_OBJ_ONLY flag is specified and the object-specific entry specified by escsi_krid_t does not exist.

CONSTRAINTS

Do not hold any locks when the ESCSI_KR_CREATE flag is specified.
Do not invoke in interrupt context when the ESCSI_KR_CREATE flag is specified.

**EXAMPLE**

```c
int errnum = ESUCCESS;
escsi_krid_t krid;
escsi_kr_key_t key;
uint32_t buf32;
uint64_t buf64;
uint32_t buf1;
char name[256];
int i, j = 0;

/* Create a /escsi/mydriver KRS node */
krid.id_type = ESCSI_NAME;
snprintf(name, 256, "/escsi/%s", "mydriver");
buf1 = (uint32_t)strlen(name);
ESCSI_ASSERT(buf1 < ESCSI_ID_NAME_SZ);
strcpy(krid.kr_info.name, name);
errno = escsi_kr_open(&krid, ESCSI_GLOBAL, ESCSI_KR_CREATE, &key);
```

**SEE ALSO**

`escsi_kr_close(9F), escsi_kr_delete(9F), escsi_kr_get_settable_attr(9F), escsi_kr_flags_t(9S), escsi_kr_key_t(9S), escsi_krid_t (9S), escsi_kr_lookup_attr(9F), escsi_kr_set_attr(9F)`
NAME

escsi_kr_set_attr - Modify the current, saved, or default value of an eSCSI attribute.

SYNOPSIS

int escsi_kr_set_attr(
    escsi_kr_key_t *key,
    char *attr_name,
    escsi_kr_flags_t flags,
    unsigned int buf_size,
    void *attr_value,
    kr_type_t attr_type
);

PARAMETERS

key        Pointer to the eSCSI KRS entry handle obtained with a call to escsi_kr_open.
attr_name  Name of the attribute.
flags      One of the following:
            ESCSI_KR_PERSISTENT  Value of the attribute is stored persistently.
            ESCSI_KR_DRVDEF     Restore the default value of the attribute.
            ESCSI_KR_DEFAULT   Value is the driver default value.
buf_size   The size of the buffer allocated for attr_value.
attr_value Attribute value.
attr_type  Attribute type.

DESCRIPTION

The escsi_kr_set_attr service sets or modifies the value of an attribute.
If the ESCSI_KR_PERSISTENT flag is set, the value of the attribute is stored persistently; otherwise, the new value is effective only until a subsequent reboot. Setting the ESCSI_KR_DEFAULT flag restores the default value of the attribute.
eSCSI drivers use the ESCSI_KR_DRVDEF flag to create default attributes at drv_install time.

RETURN VALUE

ESUCCESS   The call completed successfully.
EINVAL     The attribute name exceeds KR_NAME_LEN, or the attribute type is invalid.
ENOENT      The specified KRS node does not exist.

CONSTRAINTS

This routine must be invoked in process context only with no locks held.
There is no provision to update the value of an attribute persistently without affecting the current copy. Any update to the saved value is reflected in the current system instance.
Restoring the default value of an attribute is always persistent.
Non-persistent restoration of an attribute is not supported.

EXAMPLE

Following is an example of global attribute management:
typedef struct {
    char name[KR_NAME_LEN];      /* Attr name */
    kr_type_t type;              /* Attr type */
    uint32_t val_u32;             /* uint32 value */
    uint64_t val_u64;             /* uint64 value */
} mydriver_def_attr_t;

#define ATTR_MYDRIVER_DEFAULT_SECS   "default_secs"
/* default, min and max value of the attribute */
#define ATTR_DEF_MYDRIVER_DEFAULT_SECS 30
#define ATTR_MIN_MYDRIVER_DEFAULT_SECS 0
#define ATTR_MAX_MYDRIVER_DEFAULT_SECS xFFFFFFFF

mydriver_def_attr_t mydriver_def_glbl_attr = {
    { ATTR_MYDRIVER_DEFAULT_SECS, KR_VTYPE_UINT32,
      ATTR_DEF_MYDRIVER_DEFAULT_SECS, 0 },

int errno;
escsi_krid_t            krid;
escsi_kr_key_t          key;
uint32_t                buf;
char                    name[256];

/* Create a /escsi/"mydriver" KRS node */
krid.id_type = ESCSI_NAME;
snprintf(name, 256, "/escsi/%s", "mydriver");
strcpy(krid.kr_info.name, name);

erro = escsi_kr_open(&krid, ESCSI_GLOBAL, ESCSI_KR_CREATE, &key);
errno = escsi_kr_set_attr(&key, esc_def_glbl_attr.name,
ESCSI_KR_DRVDEF, sizeof(uint32_t),
(void *)&esc_def_glbl_attr.val_u32, KR_VTYPE_UINT32);
escsi_kr_close(&key);

Following is an example of a Device Data Repository (DDR) attribute management:

int                     errno           = ESUCCESS;
escsi_krid_t            krid;
escsi_kr_key_t          key;
uint32_t                buf1;
char                    name[256];
int    cmd_timeout;
int    cmd_timeout_def;
int    cmd_timeout_cur;
int    cmd_timeout_sav;
escsi_attr_flags_t attr_flags;

/* Create a /escsi/"mydriver" DDR entry */
krid.id_type = ESCSI_NAME;
snprintf(name, 256, "/escsi/%s", "mydriver");
strcpy(krid.kr_info.name, name);

erro = escsi_kr_open(&krid, ESCSI_GLOBAL, ESCSI_KR_CREATE, &key);

/* Create the "command timeout" attribute */
cmd_timeout = 30;
erro = escsi_kr_set_attr(&key, "command timeout",
ESCSI_KR_DRVDEF, sizeof(uint32_t),
(void *)&cmd_timeout, KR_VTYPE_UINT32);

/* Set the "command timeout" current attribute */
cmd_timeout = 20;
erro = escsi_kr_set_attr(&key, "command timeout",
ESCSI_KR_CURRENT, sizeof(uint32_t),
(void *)&cmd_timeout, KR_VTYPE_UINT32);

/* Set the “command timeout” saved attribute */
cmd_timeout = 10;
errno = escsi_kr_set_attr(&key, “command timeout”,
    ESCSI_KR_SAVED, sizeof(uint32_t),
    (void *)&cmd_timeout, KR_VTYPE_UINT32);

/* retrieve the “command timeout” attribute */
size = sizeof(uint32_t);
type = KR_VTYPE_UINT32;
errno = escsi_kr_lookup_attr(&key, “command timeout”,
    ESCSI_KR_DEFAULT, &size, &cmd_timeout, &type);

buf1 = sizeof(uint32_t);
escsi_kr_get_settable_attr(&key, “command timeout”,
    (void *)&cmd_timeout_cur,
    (void *)&cmd_timeout_def,
    (void *)&cmd_timeout_sav, &buf1,
    &attr_flags, &kr_status,
    &kr_type);

escsi_kr_close(&key);

**SEE ALSO**

**escsi_kr_close**(9F), **escsi_kr_delete**(9F), **escsi_kr_get_settable_attr**(9F), **escsi_kr_flags_t**(9S),
**escsi_kr_key_t**(9S), **escsi_kr_lookup_attr**(9F), **escsi_kr_open**(9F), **SIOC_ATTR**(9G)
NAME

escsi_krid_t - Type of the object identifier used by an eSCSI KR service.

SYNOPSIS

#include <io/escsi_node.h>

STRUCTURE MEMBERS

typedef struct {
    escsi_krid_type_t id_type;
    union {
        escsi_info_t    *info;
        escsi_addr_t    *addr;
        char            name[ESCSI_ID_NAME_SZ];
    } kr_info;
} escsi_krid_t;

id_type Specifies the eSCSI KRS identifier type.
info Pointer to an identifier of type escsi_info_t if the id_type is ESCSI_INFO.
addr Pointer to an identifier of type escsi_addr_t if the id_type is ESCSI_ADDR.
name Holds the identifier string if id_type is ESCSI_NAME.

DESCRIPTION

The escsi_krid_t structure defines the identifiers used by ESCSI KR services.

SEE ALSO

escsi_kr_flags_t(9S), escsi_kr_key_t(9S), escsi_kr_open(9F), escsi_krid_type_t(9S)
NAME
escsi_krid_type_t - Type of the object identifier used by an eSCSI KR service.

SYNOPSIS
#include <io/escsi_node.h>

STRUCTURE MEMBERS
typedef enum {
    ESCSI_ADDR,
    ESCSI_INFO,
    ESCSI_NAME
} escsi_kr_type_t;

ESCSI_ADDR The identifier is escsi_addr_t.
ESCSI_INFO The identifier is escsi_info_t.
ESCSI_NAME The identifier is a KRS path name.

DESCRIPTION
The escsi_krid_t structure defines the type of the identifier used by eSCSI KR services.

SEE ALSO
escsi_kr_flags_t(9S), escsi_kr_key_t(9S), escsi_kr_open(9F), escsi_krid_t(9S)
escsi_leg_dev_lookup(9F)

NAME
escsi_leg_dev_lookup - Convert a legacy dev into a legacy LUN

SYNOPSIS
escsi_leg_lun_t *escsi_leg_dev_lookup(
    dev_t dev
);  

PARAMETERS
dev Legacy lunpath dev_t.

DESCRIPTION
The `escsi_leg_dev_lookup` routine converts a legacy device to a legacy lunpath. It is invoked in the context of `drv_ops_t->d_open` or `drv_ops_t->d_close` entry points of legacy class drivers.

RETURN VALUES
Pointer to the legacy LUN structure.

CONSTRAINTS
The routine expects a `dev_t` corresponding to a legacy LUN.

EXAMPLE
See the `escsi_leg_lun_open(9F)` and `escsi_leg_lun_close(9F)` examples.

SEE ALSO
`escsi_leg_lun_close(9F), escsi_leg_lun_open(9F)`
escsi_leg_lun_close(9F)

NAME
escsi_leg_lun_close - Closes a legacy lun.

SYNOPSIS
void escsi_leg_lun_close(
    escsi_leg_lun_t *leg_lun,
    dev_t dev,
    int dev_type
) ;

PARAMETERS

    leg_lun Pointer to a legacy LUN structure.
    dev The devt of the legacy LUN.
    dev_type D_CHR or D_BLK device type.

DESCRIPTION
This SCSI Services public routine performs common steps in legacy LUN close. The
escsi_leg_lun_close routine is called by the legacy class drivers in the context of the
drv_ops_t->d_close entry point. It is also invoked in the context of a drv_ops_t->d_open
entry point of the legacy class driver to undo a legacy LUN open operation that failed.

RETURN VALUE
None

CONSTRAINTS
Legacy LUN semaphore is held.
Legacy LUN lock is held.
The dev_t passed must be legacy dev_t.

EXAMPLE
int mydriver_leg_close(dev_t dev, int flags, int mode)
{
    escsi_leg_lun_t *leg_lun;
    /* Convert legacy dev to legacy lun */
    leg_lun = escsi_leg_dev_lookup(dev);
    if (leg_lun == NULL) {
        return(ESUCCESS);
    }
    /*
    * Hold the legacy lun semaphore and lock
    * to protect the integrity
    * of the leg_lun-to-lpt/LUN mapping throughout the
    * entry point.
    * The legacy lun lock also protects the leg_lun
    * open counters
    */
    ESCSI_LEG_LUN_PSEMA(leg_lun);
    ESCSI_LEG_LUN_LOCK(leg_lun);
    escsi_leg_lun_close(leg_lun, dev, dev_type);
    ESCSI_LEG_LUN_UNLOCK(leg_lun);
    /*
    * mydriver_close() acquires the class driver
    */
}
* LUN semaphore and the
* LUN path queue semaphore
*/
(void)mydriver_close(dev, flags, mode);
.
.
ESCSI_LEG_LUN_VSEMA(leg_lun);
return(ESUCCESS);
}

SEE ALSO

escsi_leg_dev_lookup(9F), escsi_leg_lun_open(9F)
ESCSI_LEG_LUN_LOCK(9G)

NAME

ESCSI_LEG_LUN_LOCK - Macro to acquire the legacy LUN lock.

SYNOPSIS

ESCSI_LEG_LUN_LOCK(__leg_lun)

PARAMETERS

__leg_lun Pointer to an escsi_leg_lun_t structure.

DESCRIPTION

The ESCSI_LEG_LUN_LOCK macro acquires the legacy LUN lock. Class drivers must use the legacy LUN lock to protect the integrity of the escsi_leg_lun_t structure.

RETURN VALUES

None

CONSTRAINTS

None

EXAMPLE

escsi_leg_lun_t *leg_lun;
escsi_lun_t *lun;

ESCSI_LEG_LUN_LOCK(leg_lun);
lun = leg_lun->lun;
if (lun == NULL) {
   .
   .
   .
}
ESCSI_LEG_LUN_UNLOCK(leg_lun);

SEE ALSO

ESCSI_LEG_LUN_UNLOCK(9G)
NAME

escsi_leg_lun_open - Opens a legacy lunpath.

SYNOPSIS

int escsi_leg_lun_open(
    escsi_leg_lun_t *leg_lun,
    dev_t dev,
    int dev_type
);

PARAMETERS

leg_lun Pointer to a legacy lun structure.
dev The devt of the legacy lun.
dev_type D_CHR or D_BLK device type.

DESCRIPTION

The escsi_leg_lun_open SCSI Services public routine performs common open processing of the legacy LUN. This routine is called by the legacy class drivers drv_ops_t->d_open entry point to perform legacy opens.

RETURN VALUE

EBUSY The legacy LUN is opened exclusively by a non pass-through driver or a pass-through driver, or the legacy LUN is partially opened.
ENXIO The LUN pointer is NULL.

CONSTRAINTS

Must hold a legacy LUN semaphore.
Must hold a legacy LUN lock.
The dev_t passed must be a legacy dev_t.

EXAMPLE

int
mydriver_leg_open(dev_t dev, int oflags, intptr_t dummy, int mode)
{
    escsi_lun_t *lun;
    escsi_leg_lun_t *leg_lun;

    /* If legacy mode is disabled on the platform, fail the open. */
    if (io_get_legacy_mode() == FALSE) {
        return ENXIO;
    }

    leg_lun = escsi_leg_dev_lookup(dev);
    if (leg_lun == NULL) {
        return(ENXIO);
    }

    /* hold the legacy lun semaphore to protect the integrity
       of the leg_lun-to-lpt/LUN mapping throughout the
       entry point and to single-thread open/close.
    */
    ESCSI_LEG_LUN_PSEMA(leg_lun);
    ESCSI_LEG_LUN_LOCK(leg_lun);
lun = leg_lun->lun;
if (lun == NULL) {
    ESCSI_LEG_LUN_UNLOCK(leg_lun);
    ESCSI_LEG_LUN_VSEMA(leg_lun);
    return(ENXIO);
}

errno = escsi_leg_lun_open(leg_lun, dev, dev_type);
if (errno != ESUCCESS) {
    ESCSI_LEG_LUN_UNLOCK(leg_lun);
    ESCSI_LEG_LUN_VSEMA(leg_lun);
    return(errno);
}

if (ESCSI_LEG_DD_OPEN_CNT(leg_lun) > 1) {
    ESCSI_LEG_LUN_UNLOCK(leg_lun);
    ESCSI_LEG_LUN_VSEMA(leg_lun);
    return(ESUCCESS);
}

ESCSI_LEG_LUN_UNLOCK(leg_lun);
	/*
	* Forward first open on legacy lun to new driver
	* mydriver_open() internally takes and releases the
	* class driver LUN semaphore, and the LUN path
	* queue semaphore.
	*/
errno = mydriver_open(dev, oflags, dummy, mode);
if (errno != ESUCCESS) {
    ESCSI_LEG_LUN_UNLOCK(leg_lun);
    /* Undo legacy open counts */
    escsi_leg_lun_close(leg_lun, dev, dev_type);
    ESCSI_LEG_LUN_UNLOCK(leg_lun);
}

ESCSI_LEG_LUN_VSEMA(leg_lun);
return(errno);
}

SEE ALSO

escsi_leg_dev_lookup(9F), escsi_leg_lun_close(9F)
NAME

escsi_leg_lun_t - Legacy LUN data structure.

SYNOPSIS

#include <io/escsi_stack.h>

STRUCTURE MEMBERS

lock        Synchronize legacy LUN access.
ki_handle   Kmetrics handle.
lpt         Pointer to an escsi_lpt_t structure.
lun         Pointer to an escsi_lun_t structure.
flags       Flags defined in escsi_leg_lun_flags_t.
basis       Legacy bus instance.
tgt_id      Legacy target id.
lun_id      Legacy LUN ID.
open_cnt    Legacy LUN open count.
raw_open_cnt Legacy LUN raw open count.
sctl_open_cnt Legacy LUN pass-through open count.
open_status Can be used by the class drivers to cache the open status. For example, save
              the status of the path open on the lunpath corresponding to the legacy
              LUN.
sema        Legacy LUN eSCSI semaphore to serialize open or close on the legacy dsf.
leg_tgt     Pointer to a legacy target escsi_leg_tgt_t structure.
lpt_qelem   Queue element in the escsi_lpt_t->leg_lun_q queue.
stat        Reserved.

DESCRIPTION

The escsi_leg_lun_t structure defines a legacy LUN data structure. It corresponds to a legacy
lunpath.

SEE ALSO
**ESCSI_LEG_LUN_PSEMA(9G)**

**NAME**

ESCSI_LEG_LUN_PSEMA - Macro to acquire the legacy LUN semaphore.

**SYNOPSIS**

ESCSI_LEG_LUN_PSEMA(__leg_lun)

**PARAMETERS**

__leg_lun Pointer to an escsi_leg_lun_t structure.

**DESCRIPTION**

The ESCSI_LEG_LUN_PSEMA macro acquires the legacy LUN semaphore. The legacy LUN semaphore protects the integrity of the legacy LUN to lunpath or LUN mapping throughout legacy class drivers d_open or d_close entry points.

**RETURN VALUES**

None

**CONSTRAINTS**

Legacy class drivers must hold the legacy LUN semaphore and lock to protect the integrity of the legacy LUN to lunpath or LUN mapping throughout legacy class drivers d_open or d_close entry points.

The legacy LUN semaphore must be held before the class driver LUN semaphore.

**EXAMPLE**

```c
int mydriver_leg_open(dev_t dev, int oflags, intptr_t dummy, int mode)
{
    escsi_lun_t             *lun;
    escsi_leg_lun_t         *leg_lun;
    int                     errno;

    /* Map legacy dev to leg lun */
    leg_lun = escsi_leg_dev_lookup(dev);

    /*
     * hold the legacy lun semaphore and lock to
     * protect the integrity
     * of the leg_lun-to-lpt/LUN mapping throughout
     * the entry point
     */
    ESCSI_LEG_LUN_PSEMA(leg_lun);

    .
    .
    .
    ESCSI_LEG_LUN_VSEMA(leg_lun);
    return(errno);
}
```

**SEE ALSO**

ESCSI_LEG_LUN_VSEMA(9G)
ESCSI_LEG_LUN_UNLOCK(9G)

NAME
ESCSI_LEG_LUN_UNLOCK - Macro to release the legacy LUN lock.

SYNOPSIS
ESCSI_LEG_LUN_UNLOCK(__leg_lun)

PARAMETERS
__leg_lun Pointer to an escsi_leg_lun_t structure.

DESCRIPTION
The ESCSI_LEG_LUN_UNLOCK macro releases the legacy LUN lock. Class drivers must use the
legacy LUN lock to protect the integrity of the escsi_leg_lun_t structure.

RETURN VALUES
None

CONSTRAINTS
None

EXAMPLE
See the ESCSI_LEG_LUN_LOCK (9G) example.

SEE ALSO
ESCSI_LEG_LUN_LOCK(9G)
ESCSI_LEG_LUN_VSEMA(9G)

NAME

ESCSI_LEG_LUN_VSEMA - Macro to release the legacy LUN semaphore.

SYNOPSIS

ESCSI_LEG_LUN_VSEMA(__leg_lun)

PARAMETERS

__leg_lun Pointer to an escsi_leg_lun_t structure.

DESCRIPTION

The ESCSI_LEG_LUN_VSEMA macro releases the legacy LUN semaphore. The legacy LUN semaphore protects the integrity of the legacy LUN to lunpath or LUN mapping throughout legacy class drivers d_open or d_close entry points.

RETURN VALUES

None

CONSTRAINTS

Legacy class drivers must hold the legacy LUN semaphore and lock to protect the integrity of the legacy LUN to lunpath or LUN mapping throughout legacy class drivers d_open or d_close entry points.

The legacy LUN semaphore must be held before the class driver LUN semaphore.

EXAMPLE

See the ESCSI_LEG_LUN_PSEMA(9G) example.

SEE ALSO

ESCSI_LEG_LUN_PSEMA(9G)
NAME

escsi_leg_strategy - Process I/O issues on legacy LUN.

SYNOPSIS

int escsi_leg_strategy(
    struct buf *bp
);

PARAMETERS

bp Pointer to buf structure.

DESCRIPTION

The escsi_leg_strategy routine is a generic SCSI Services interface that processes I/O requests issued on a legacy lunpath DSF. This routine performs the preprocessing of legacy I/O requests sent on legacy lunpath DSFs and forwards the I/O requests to the new escsi_strategy routine.

Legacy class drivers issuing block I/O requests on a legacy lunpath DSF use escsi_strategy as the driver drv_ops_t->d_strategy entry point. Legacy class drivers issuing raw I/O requests on a legacy lunpath DSF using the drv_ops_t->d_read and drv_ops_t->d_write entry points can also invoke escsi_strategy using the physio routine.

RETURN VALUE

Always returns 0 (zero).

CONSTRAINTS

No locks held around this call.

Must be the drv_ops_t->d_strategy entry point for all legacy class drivers.

Expected to be strategy entry point for all legacy class drivers.

EXAMPLE

static drv_ops_t mydriver_ops = {
    mydriver_open, /* Legacy d_open entry point */
    mydriver_close, /* Legacy d_close entry point */
    escsi_leg_strategy, /* Legacy d_strategy entry point */
    NULL, /* Legacy d_dump entry point */
    NULL, /* d_psize entry point */
    NULL, /* Reserved */
    mydriver_read, /* Legacy d_read entry point */
    mydriver_write, /* Legacy d_write entry point */
    NULL, /* Legacy d_ioctl entry point */
    NULL, /* Legacy d_select entry point */
    NULL, /* Legacy d_option1 entry point */
    NULL, /* pfilter entry point */
    NULL, /* d_psize1 entry point */
    NULL, /* Reserved */
    NULL, /* Legacy d_aio_ops entry point */
    (C_ALLCLOSES | C_MGR_IS_MP) /* Legacy d_flags */
};
mydriver_read(dev_t dev, struct uio *uio)
{
    
    error = physio(escsi_leg_strategy, (struct buf *) NULL, dev, B_READ,
                   mydriver_minphys, uio)

};
mydriver_write(dev_t dev, struct uio *uio)
{
    .
    .
    error = physio(escsi_leg_strategy, (struct buf *) NULL, dev, B_WRITE,
                    mydriver_minphys, uio);
    .
    .
    return (error);
}

SEE ALSO

escsi_strategy(9F)
NAME
escsi_lookup_obj - Look up a SCSI object.

SYNOPSIS

```c
escsi_info_t *escsi_lookup_obj(
    escsi_id_info_t *id
);
```

PARAMETERS

`id` Pointer to an `escsi_id_info_t` generic structure describing a SCSI object involved in a management ioctl.

DESCRIPTION

The `escsi_lookup_obj` routine returns a pointer to an `escsi_info_t` structure corresponding to the SCSI object whose identifier is passed in. The routine is typically invoked by class drivers when implementing management ioctls.

RETURN VALUE

Pointer to `escsi_info_t` corresponding to the SCSI object whose identifier is passed in.

CONSTRAINTS

This function internally performs an `ESCSI_HOLD` with flag set to `ESCSI_TRUE` to protect against object deletion while the ioctl is processed. The caller must perform a `ESCSI_RELE` on the object. Do not hold locks around this call.

EXAMPLE

See the `SIOC_ATTR(9G)` and `SIOC_CLEAR_STAT(9G)` examples.

SEE ALSO

`ESCSI_HOLD(9G), ESCSI_RELE(9G), SIOC_ATTR(9G), SIOC_CLEAR_STAT(9G)`
ESCSI_LPT_LOCK(9G)

NAME
ESCSI_LPT_LOCK - Macro to acquire the lunpath lock.

SYNOPSIS
ESCSI_LPT_LOCK(__lpt)

PARAMETERS
__lpt    Pointer to an escsi_lpt_t structure.

DESCRIPTION
The ESCSI_LPT_LOCK macro acquires the lunpath lock. Class drivers must use the lunpath lock to protect the integrity of the escsi_lpt_t structure.

RETURN VALUES
None

CONSTRAINTS
The SCSI lock order of acquisition is:
1. LUN lock
2. lunpath

EXAMPLE
See the ESCSI_LUN_LOCK(9G) example.

SEE ALSO
ESCSI_IS_LPT_ONLINE(9G), ESCSI_LPT_UNLOCK(9G), ESCSI_LUN_LOCK(9G)
NAME
escsi_lpt_offline - Take a lunpath offline.

SYNOPSIS
void escsi_lpt_offline(
    escsi_lpt_t *lpt
);

PARAMETERS
lpt Pointer to the lunpath to be taken offline.

DESCRIPTION
The escsi_lpt_offline routine performs some generic operations to take the lunpath offline. It is invoked by class drivers in the context of the escsi_ddsw_t->dd_aen entry point as the first step to handle an ESCSI_LPT_OFFLINE event.

RETURN VALUE
None

CONSTRAINTS
Do not hold any locks.

EXAMPLE
See the dd_aen(9E) example.

SEE ALSO
dd_aen(9E)
escsi_lpt_online(9F)

NAME
escsi_lpt_online -- Bring a lunpath back online.

SYNOPSIS
void escsi_lpt_online(
  escsi_lpt_t *lpt
);

PARAMETERS
lpt Pointer to the lunpath previously taken offline.

DESCRIPTION
The escsi_lpt_offline routine performs some generic operations to bring a lunpath back online. It is invoked by class drivers in the context of the escsi_ddsw_t->dd_aen entry point as the first step to handle an ESCSI_LPT_ONLINE event.

RETURN VALUE
None

CONSTRAINTS
Do not hold any locks.

EXAMPLE
See the dd_aen(9E) example.

SEE ALSO
dd_aen(9E)
escsi_lpt_open(9F)

NAME

escsi_lpt_open – Opens a lunpath.

SYNOPSIS

void escsi_lpt_open(
    escsi_lpt_t *lpt,
    escsi_optype_t waitok,
    int oflags,
    escsi_cbfn_t *cbfn,
    int *errnop
);

PARAMETERS

lpt Pointer to a escsi_lpt_t structure.
waitok Set to ESCSI_WAITOK or ESCSI_NOWAIT. Caller can specify whether the lunpath open must be blocking or non-blocking. In the non-blocking case, the lunpath open call must still be made in a block-able context so as to facilitate resource acquisition.
oflags Open flags from the d_open call.
cbfn Pointer to escsi_cbfn_t, which contains a cbfunc and a cb_arg. Only applicable when waitok equals ESCSI_NOWAIT.
errnop Pointer to errno. Completion status of lunpath open.

DESCRIPTION

The escsi_lpt_open SCSI Services routine performs generic lunpath open activity. It sends the following commands onto the lunpath: INQUIRY, INQUIRY vpd page 0x80, and INQUIRY vpd page 0x83. This routine must be called by the class driver for each lunpath being opened in the context of LUN open.

The class driver can invoke the service either synchronously with the waitok parameter set to ESCSI_WAITOK, or asynchronously with the waitok parameter set to ESCSI_NOWAIT. In the latter case, when the lunpath open operation asynchronously completes, SCSI Services invokes the cbfn->cb_func callback function with the cbfn->cb_arg parameter.

RETURN VALUE

None

CONSTRAINTS

No locks can be held around this call. The lpt->lun element might not be set and must not be used in this function.
Caller must supply a lpt.
Caller must specify if the call is blocking or non-blocking.
Caller must provide a cbfn if the call is non-blocking.
Caller cannot be on ICS (even for non-blocking lunpath opens).
ESCSI_NOWAIT calls to escsi_lpt_open always return the completion status through the supplied cbfn->cb_func second argument.

EXAMPLE

Following is an example of an asynchronous call to escsi_lpt_open:
void mydriver_lpt_open(escsi_lpt_t *lpt, int oflags)
{
    escsi_cdbinfo_t *cdbinfo;
    escsi_cbfn_t cbfn;
    int rval;
.
.
    cdbinfo = escsi_cdbinfo_alloc(NODEV,
        D_CHR, NULL, lpt, NULL, ESCTL_IO_LPT,
        ESCTL_IO_NOWAIT, 0, (uint8_t *)NULL, 0,
        (uint8_t *)NULL, 0, (escsi_status_action_t *)NULL,
        0, (escsi_cbfunc_t) NULL, NULL, NULL, M_WAITOK);

    ESCSI_SET_CBFN(&cbfn, mydriver_lpt_open_cbfn, cdbinfo);
    escsi_lpt_open(lpt, ESCSI_NOWAIT, oflags, &cbfn, &rval);
    return
}
/* SCSI Services asynchronously invokes mydriver_lpt_open() when
 * the lunpath open completes. */
void mydriver_lpt_open_cbfn(void *cdbinfo_hdl, int errno)
{
    escsi_cdbinfo_t *cdbinfo = (escsi_cdbinfo_t*)cdbinfo_hdl;
    escsi_lpt_t *lpt = (escsi_lpt_t *)lprb_cdbinfo->lpt;
.
.
    /* errno holds the escsi_lpt_open() completion status */
    if (errno != ESUCCESS) {
        /* The generic lunpath open operation failed */
        return;
    }

    /* The generic lunpath open operation succeeded.
     * Perform the class driver specific lunpath open
     * operation. */
    ...
.
    return;
}
Following is an example of a synchronous call to escsi_lpt_open:
int
mydriver_lpt_open(escsi_lpt_t *lpt, uint64_t options, int oflags)
{
    int error;

    escsi_lpt_open(lpt, ESCSI_WAITOK, oflags, NULL, &error);
    if (error != ESUCCESS) {
        return error;
    }

    /* The generic lunpath open operation succeeded.
     * Perform the class driver specific lunpath open
     * operation. */
    ...
.
.
    /* Return the errno status of ESUCCESS. */
return ESUCCESS;
}

SEE ALSO
escsi_cdbinfo_alloc(9F), ESCSI_SET_CBFN(9G)
NAME
escsi_lpt_t - Common SCSI lunpath data structure.

SYNOPSIS
#include <io/escsi_lun.h>

DESCRIPTION
This structure represents a lunpath object.

STRUCTURE MEMBERS

```
struct escsi_lpt {
    escsi_info_t          info;
    void               *node;
    escsi_list_t         lpt_hash_qelm;
    escsi_list_t         path_qelm;
    escsi_list_t         evt_shdw_qelm;
    escsi_list_t         nexus_q;
    escsi_thcb_t         thcb;
    escsi_thcb_t         prb_thcb;
    escsi_cbfn_t         prb_cbfn;
    escsi_lptfail_rsn_t  offline_rsn;
    escsi_wwid_type_t    prev_wwid_type;
    escsi_wwid_type_t    wwid_type;
    escsi_wwid_code_set_t wwid_code_set;
    escsi_wwid_assoc_t   wwid_assoc;
    struct iovec            prev_wwid;
    struct iovec            wwid;
    scsi_serial_t        serial;
    uint32_t             serial_len;
    escsi_lpt_state_t    prev_state;
    escsi_spoc_state_t   spoc_state;
    escsi_suspend_rsn_t  suspend_rsn;
    int                active_ios;
    int               n_scbs;
    int                max_msecs;
    uint32_t          max_io_size;
    uint32_t          inquiry_len;
    escsi_nexus_t      nexus;
    escsi_lpt_stat_t   stat;
    escsi_io_stat_t    io_stat;
};
```

info                   Specifies common fields.

node                   Pointer to lunpath I/O node.

lpt_hash_qelm          Linked list element used to enqueue lunpath onto the lunpath hash table of the parent target path.

path_qelm              Linked list element used to enqueue lunpath onto lun->path_q.
**evt_shdw_qelm**: Linked list element used to enqueue the evt shadow queue.

**nexus_q**: Linked list of escsi_scb structures waiting for an I-T-L-Q nexus. The queue depth is at maximum and these I/Os are flow controlled.

**thcb**: Thread control block used to enqueue lunpath manager requests to the worker thread.

**prb_thcb**: Thread control block used to enqueue this lunpath onto the controller target path scan_q during lunpath probe. Used by the lpb_fsm (lunpath probe FSM).

**prb_cbfn**: Specifies the callback function to be called on completion of lunpath probe. Used by lpt_fsm.

**prb_state**: Specifies the state of the lunpath probe FSM.

**prb_status**: The errno completion status of the last lunpath probe.

**apsw**: Pointer to active-passive switch table entry for the Active-Passive Switch managing this LUN. This is NULL for a symmetric access device.

**tp**: Back pointer to the target path.

**lun**: Back pointer to the LUN.

**leg_lun**: Back pointer to the legacy LUN.

**if_lpt**: Interface driver-specific lunpath handle returned on completion of if_lpt_open.

**dd_lpt**: Class driver-specific lunpath handle.

**ulp_iobj**: Back pointer to stack-specific iobj.

**tagpool_ptr**: Pointer to a pointer to the tag stack being used. The tag stack might be allocated on a per iobj, target path, or lunpath basis as requested by the interface driver.

**tagpool**: Pointer to the lunpath tag stack, if the interface driver has requested tag allocation on a per lunpath basis.

**offline_rsn**: Specifies the reason the lunpath is offline.

**prev_wwid_type**: ID type from a previous EVPD page 0x83 inquiry.

**wwid_type**: ID type from the most recent EVPD page 0x83 inquiry.

**wwid_code_set**: Code set from the most recent EVPD page 0x83 inquiry.

**wwid_assoc**: EVPD inquiry page 0x83 descriptor association.

**prev_wwid**: Previous LUN world wide identifier.

**wwid**: Current LUN world wide identifier.

**serial**: EVPD page 0x80

**serial_len**: Length of EVPD page 0x80

**prev_state**: Previous lunpath state.

**spoc_state**: Open or close state of lunpath from last spoc execution.

**suspend_rsn**: Set of ORed reason bits to indicate the various reasons for the lunpath suspension.

**active_ios**: Number of I/Os currently outstanding on this lunpath.

**n_scbs**: Number of SCBs currently in use on this lunpath.

**max_msecs**: Maximum I/O timeout. This is the class driver-specific timeout added to the interface driver additional latency. A default value is assigned when the lunpath is first allocated.

**max_io_size**: Maximum allowed I/O size on the LUN to which this lunpath belongs.
inquiry_len  Length of the inquiry data.
inquiry_data Inquiry data obtained on last lunpath open.
nexus I/OP Credit Manager nexus related fields.
stat Common lunpath statistics.
io_stat I/O statistics.

CONSTRAINTS
None

SEE ALSO
NAME
escsi_lpt_uninit - Uninitializes a lunpath data structure.

SYNOPSIS
void escsi_lpt_uninit(
    escsi_lpt_t *lpt
);

PARAMETERS
lpt Pointer to the lunpath structure.

DESCRIPTION
The escsi_lpt_offline routine deallocates the common escsi_lpt_t resources and waits
for all I/O requests using the lunpath to complete. It is invoked in the context of a driver
dd_lpt_uninit entry point.

RETURN VALUE
None

CONSTRAINTS
Must be called in a blocking context.
Do not hold any locks.
A LUN semaphore must be held around this call.

EXAMPLE
void
mydriver_dd_lpt_uninit(escsi_lpt_t *lpt)
{
    /* LUN semaphore is held by calling SCSI Services */

    /* Perform class driver specific lunpath
    * uninitialization
    */
    .
    .
    .
    /* Perform common lunpath uninitialization */
    escsi_lpt_uninit(lpt);
    return;
}

SEE ALSO
dd_lpt_init(9E), dd_lpt_uninit(9E), escsi_new_lpt(9F)
ESCSI_LPT_UNLOCK(9G)

NAME
ESCSI_LPT_UNLOCK - Macro to release the lunpath lock.

SYNOPSIS
ESCSI_LPT_UNLOCK(__lpt)

PARAMETERS
__lpt  Pointer to an escsi_lpt_t structure.

DESCRIPTION
The ESCSI_LPT_UNLOCK macro releases the lunpath lock. Class drivers must use the lunpath lock to protect the integrity of the escsi_lpt_t structure.

RETURN VALUES
None

CONSTRAINTS
The SCSI lock order of acquisition is:
1. LUN lock
2. lunpath

EXAMPLE
See the ESCSI_LUN_LOCK(9G) example.

SEE ALSO
ESCSI_IS_LPT_ONLINE(9G), ESCSI_LPT_LOCK(9G), ESCSI_LUN_LOCK(9G)
NAME
escsi_lun_close - Close a lun

SYNOPSIS
void escsi_lun_close(
    escsi_lun_t *lun,
    dev_t dev,
    int dev_type
);

PARAMETERS
lun Pointer to escsi_lun structure.
dev The devt (legacy or new) of the LUN being closed.
dev_type D_BLK or D_CHR.

DESCRIPTION
The escsi_lun_close SCSI Services routine performs generic LUN close activity. This routine
must be called from the class driver close entry point as the first step in the d_close entry point.
It looks up the dev_t, decrements the open count, and returns the LUN pointer.

RETURN VALUE
None

CONSTRAINTS
LUN semaphore is held by the caller.
No locks to be held around this call.
Called in a blocking context.
The class driver calls these functions, which return control back to the class driver. This allows
the open and close of lunpaths to be either asynchronous or parallel, or synchronous or serialized.

EXAMPLE
int
mydriver_close(dev_t dev, int flags, int mode)
{
    escsi_lun_t     *lun = NULL;
    escsi_lpt_t     *lpt = NULL;
    int             errno = ESUCCESS;
    /* Lookup lun dev_t. */
    lun = escsi_devt_to_lun(dev, D_CHR);
    if(lun == NULL) {
        return ENXIO;
    }
    /* Acquire class driver LUN sema to single thread driver
     * entry points.
     */
    ESCSI_LUN_DD_PSEMA(lun);
    /*
     * Acquire LUN path queue sema to protect integrity
     * of lun->path_q during close
    
    1010 SCSI Reference Pages
ESC_LUN_PSEMA(lun);

escsi_lun_close(lun, dev, D_CHR);
/* Class driver LUN close specific operations */
.
.
return errno;
}

SEE ALSO

escsi_dev_to_lun(9F)
NAME

escsi_lun_compute_path_cnt - Compute standby and failed path counts of a LUN.

SYNOPSIS

void escsi_lun_compute_path_cnt(
    escsi_lun_t *lun
);

PARAMETERS

lun    Pointer to an escsi_lun structure.

DESCRIPTION

The escsi_leg_dev_lookup routine computes standby and failed path counts of a LUN.

RETURN VALUES

None

CONSTRAINTS

The caller must hold a LUN lock.

EXAMPLE

See the escsi_enq_active_lpt(9F) example.

SEE ALSO

escsi_enq_active_lpt(9F)
ESCSI_LUN_DD_PSEMA(9G)

NAME
ESCSI_LUN_DD_PSEMA - Macro to acquire the class driver LUN semaphore.

SYNOPSIS
ESCSI_LUN_DD_PSEMA(__lun)

PARAMETERS
__lun Pointer to an escsi_lun_t structure.

DESCRIPTION
The ESCSI_LUN_DD_PSEMA macro attempts to acquire the class driver LUN semaphore (escsi_lun_t->dd_sema). If the semaphore is not acquired, the macro waits until the semaphore is available.

Class drivers can use this LUN semaphore to single-thread their drv_ops_t entry points with the pass-through esctl driver entry points. Serial class drivers can use this to single-thread their I/Os with any pass-through I/O concurrently sent to a device.

The class driver LUN semaphore must be acquired before the LUN semaphore and must be released after the LUN semaphore.

RETURN VALUES
None

CONSTRAINTS
The SCSI semaphore order of acquisition is:
1. Class driver LUN semaphore
2. LUN semaphore

EXAMPLE
int
mydriver_open(dev_t dev, int oflags, intptr_t dummy, int mode)
{
    escsi_lun_t *lun;

    /* Lookup lun dev_t. */
    lun = escsi_devt_to_lun(dev, dev_type);

    /*
     * Acquire class driver lun sema to single-thread
     * driver entry points with esctl driver entry
     * points
     */
    ESCSI_LUN_DD_PSEMA(lun);

    /*
     * Acquire LUN path queue sema to protect
     * the integrity of lun->path_q
     * during open
     */
    ESCSI_LUN_PSEMA(lun);
    errno = escsi_lun_open(lun, dev, D_CHR);
    .
    .
    .
    /* Open every lunpath in the lun->path_q */
    .
ESCSI_LUN_VSEMA(lun);
ESCSI_LUN_DD_VSEMA(lun);
return ESUCCESS;
}

SEE ALSO

ESCSI_LUN_DD_VSEMA(9G), ESCSI_LUN_VSEMA(9G)
ESCSI_LUN_DD_VSEMA(9G)

NAME
ESCSI_LUN_DD_VSEMA - Macro to release the class driver LUN semaphore.

SYNOPSIS
ESCSI_LUN_DD_VSEMA(__lun)

PARAMETERS
__lun Pointer to an escsi_lun_t structure.

DESCRIPTION
The ESCSI_LUN_DD_VSEMA macro releases the class driver LUN semaphore.

RETURN VALUES
None

CONSTRAINTS
The class driver LUN semaphore must be released after the LUN semaphore.

EXAMPLE
See the ESCSI_LUN_DD_PSEMA(9G) example.

SEE ALSO
ESCSI_LUN_DD_PSEMA(9G)
ESCSI_LUN_LOCK(9G)

NAME

ESCSI_LUN_LOCK -- Macro to acquire the LUN lock.

SYNOPSIS

ESCSI_LUN_LOCK(__lun)

PARAMETERS

__lun Pointer to an escsi_lun_t structure.

DESCRIPTION

The ESCSI_LUN_LOCK macro acquires the LUN lock. Class drivers must use the LUN lock to protect the integrity of the escsi_lun_t structure.

RETURN VALUES

None

CONSTRAINTS

The SCSI lock order of acquisition is:
1. LUN lock
2. lunpath

EXAMPLE

escsi_lun_t *lun;
escsi_list_t *next_elm;
escsi_lpt_t *lpt;

/* Acquire the LUN lock to access the lun->path_q */
ESCSI_LUN_LOCK(lun);
next_elm = ESCSI_NEXT(&lun->path_q);
lpt = ESCSI_GET_PARENT(next_elm);

ESCSI_LPT_LOCK(lpt);
if (!(ESCSI_IS_LPT_ONLINE(lpt))) {
    /* Do something on the lunpath. */
}
ESCSI_LPT_UNLOCK(lpt);

ESCSI_LUN_UNLOCK(lun);

SEE ALSO

ESCSI_LPT_LOCK(9G), ESCSI_LPT_UNLOCK(9G), ESCSI_LUN_UNLOCK(9G)
escsi_lun_open(9F)

NAME
escsi_lun_open - Open a lun.

SYNOPSIS
int escsi_lun_open(
        escsi_lun_t *lun,
        dev_t dev,
        int dev_type
    );

PARAMETERS
lun Pointer to the escsi_lun_t structure to be opened.
dev The devt (legacy or new) of the LUN being opened.
dev_type D_BLK or D_CHR

DESCRIPTION
The escsi_lun_open SCSI service is the common LUN open routine. This routine must be
called by the class driver drv_ops_t->d_open entry point as one of the first steps. A lun open
can fail for the following reasons:
• The LUN is not bound to any class driver. A LUN is bound to a class driver during the
  process of discovering the LUN devices.
• The LUN is suspended.
• A pass-through exclusive open is pending on the LUN.
• The LUN was already opened in exclusive mode by the driver (LF_EXCLUSIVE and
  LF_DISABLE_OPENS flags are set).

RETURN VALUE
errno errno completion status of common LUN open code.

CONSTRAINTS
LUN semaphore is held.
No locks to be held around this call.
Called in a blocking context.

EXAMPLE
int
mydriver_open(dev_t dev, int oflags, intptr_t dummy, int mode)
{
    escsi_lun_t     *lun;
    escsi_ipt_t     *ipt;

    lun = escsi_devt_to_lun(dev, dev_type);
    if(lun == NULL) {
        return ENXIO;
    }
    ESCSI_LUN_PSEMA(lun);

    errno = escsi_lun_open(lun, dev, D_CHR);
    if (errno != ESUCCESS) {
        ESCSI_LUN_VSEMA(lun);
        return errno;
    }
/ * Process class driver specific open */

if (error condition) {
    escsi_lun_open_failed(lun, dev, dev_type);
    ESCSI_LUN_VSEMA(lun);
    return EACCES;
}

ESCSI_LUN_VSEMA(lun);
return ESUCCESS;

**SEE ALSO**

escsi_devt_to_lun(9F), escsi_lun_open_failed(9F), ESCSI_LUN_PSEMA(9G), ESCSI_LUN_VSEMA(9G)
escsi_lun_open_failed(9F)

NAME
escsi_lun_open_failed - Undo the actions of escsi_lun_open on failure.

SYNOPSIS
int escsi_lun_open_failed(  
    escsi_lun_t *lun,  
    dev_t dev,  
    int dev_type  
);  

PARAMETERS
lun Pointer to escsi_lun structure.
dev The devt (legacy or new) of the LUN being opened.
dev_type D_BLK or D_CHR

DESCRIPTION
On open failure, this function undoes the actions of escsi_lun_open.

RETURN VALUE
None

CONSTRAINTS
Must hold a LUN semaphore.
Do not hold any locks around this call.
Called in a blocking context.

EXAMPLE
int
mydriver_open(dev_t dev, int oflags, intptr_t dummy, int mode)  
{  
    escsi_lun_t     *lun;  
    escsi_lpt_t     *lpt;
    lun = escsi_devt_to_lun(dev, dev_type);
    if(lun == NULL)  
    {  
        return ENXIO;
    }
    ESCSI_LUN_PSEMA(lun);
    errno = escsi_lun_open(lun, dev, D_CHR);
    if (errno != ESUCCESS)  
    {  
        ESCSI_LUN_VSEMA(lun);
        return errno;
    }
    /* Process class driver specific open */
    .
    .
    .
    if (error condition)  
    {  
        escsi_lun_open_failed(lun, dev, dev_type);
        ESCSI_LUN_VSEMA(lun);
        return EACCES;
    }
ESCSI_LUN_VSEMA(lun);
return ESUCCESS;
}

SEE ALSO
escsi_devt_to_lun(9F), escsi_lun_open(9F), ESCSI_LUN_PSEMA(9G), ESCSI_LUN_VSEMA(9G)
ESCSI_LUN_PSEMA(9G)

NAME

ESCSI_LUN_PSEMA - Macro to acquire the LUN semaphore.

SYNOPSIS

ESCSI_LUN_PSEMA(__lun)

PARAMETERS

__lun Pointer to an escsi_lun_t structure.

DESCRIPTION

The ESCSI_LUN_PSEMA macro acquires the LUN semaphore. If the semaphore is not acquired, the macro waits until the semaphore is available.

Class drivers must use the LUN semaphore to protect the integrity of the lun->path_q in the context of the d_open and d_close entry points.

RETURN VALUES

None

CONSTRAINTS

The SCSI semaphore order of acquisition is:
1. Class driver LUN semaphore
2. LUN semaphore

EXAMPLE

int
mydriver_open(dev_t dev, int oflags, intptr_t dummy, int mode)
{
    escsi_lun_t *lun;

    /* Lookup lun dev_t. */
    lun = escsi_devt_to_lun(dev, dev_type);

    /*
    * Acquire LUN path queue sema to protect
    * the integrity of lun->path_q
    * during open
    */
    ESCSI_LUN_PSEMA(lun);
    errno = escsi_lun_open(lun, dev, D CHR);
    .
    .
    /* Open every lunpath in the lun->path_q */
    .
    .
    ESCSI_LUN_VSEMA(lun);
    return ESUCCESS;
}

SEE ALSO

ESCSI_LUN_DD_PSEMA(9G), ESCSI_LUN_VSEMA(9G)
escsi_lun_set_health(9F)

NAME
escsi_lun_set_health - Set the health of a LUN I/O node.

SYNOPSIS

void escsi_lun_set_health(
   escsi_lun_t *lun
);

PARAMETERS
lun Pointer to an escsi_lun_t structure.

DESCRIPTION

The escsi_lun_set_health routine sets the I/O node health of the LUN based on its current internal state.

If the LUN is suspended, the health is set to disabled.
If the LUN load balancing policy is ESCSI_LPT_LOCKDOWN and there is no lunpath available (lun->next_lpt is NULL), the health is set to offline.
If all lunpaths of the LUN are offline, the health is set to offline.
If none of the preceding conditions are encountered, the LUN health is set to online.
A system administrator can see the health of all disk and tape LUN I/O nodes by running the following commands, respectively:

# ioscans -P health -C disk
.
.
# ioscans -P health -C tape
.
.

RETURN VALUES
None

CONSTRAINTS
Do not hold any locks around this call.

EXAMPLE

See the dd_aen(9F) example.

SEE ALSO

dd_aen(9F), escsi_lun_t(9S)
NAME

escsi_lun_t - Common SCSI lun data structure.

SYNOPSIS

#include <io/escsi_stack.h>

DESCRIPTION

The escsi_lun_t structure represents a SCSI LUN.

STRUCTURE MEMBERS

struct escsi_lun {
    escsi_info_t info;
    int sema;
    void *node;
    char lpt_desc[ESCSI_MAX_LPT_DESC_LEN];
    *devopt_arr;
    escsi_list_t path_q;
    thcb;
    escsi_lpt_t *next_lpt;
    *transient_q;
    struct buf *special_q;
    struct iovec wwid;
    struct escsi_wwid_code_set_t wwid_code_set;
    struct escsi_wwid_assoc_t wwid_assoc;
    struct escsi_wwid_type_t wwid_type;
    struct scsi_serial serial;
    int open_cnt;
    int raw_open_cnt;
    int esctl_open_cnt;
    int16_t total_path_cnt;
    int16_t act_path_cnt;
    int16_t standby_path_cnt;
    int16_t failed_path_cnt;
    uint32_t max_io_size;
    uint32_t pref_io_size;
    uint64_t io_id;
    int active_ios;
    uint64_t lid;
    dev_t dev;
    int dev_type;
    int16_t load_bal_policy;
    uint32_t transient_secs;
    clock_t lbolt_online_exit;
    union sense_data
        sense_bytes;
        num_trc_recs;
        *trc_buf_t;
        *trc_info;
        *esctl;
        *esctl_wwid;
        *esctl_lun;
        *esctl_lun_stat_t;
    } /* struct escsi_lun */
info
  Specifies a structure containing fields common to all structures. This
  is used for controller, target path, lunpath, and LUN.

sema
  Specifies the LUN eSCSI semaphore.

node
  Pointer to the LUN I/O node handle.

lpt_desc
  Specifies the buffer for holding the description property of all lunpaths
  of this LUN. This buffer can be referenced by the I/O infrastructure
  as a desc property, and is registered with the P_NO_COPY flag.

devopt_arr
  Specifies the LUN minor number table.

path_q
  Specifies the queue of all lunpaths to this LUN.

active_path_q
  Specifies the queue of active, usable lunpaths to this LUN.

thcb
  Specifies the callback structure to store the function and argument
  to be called on a thread.

next_lpt
  Pointer to the next lunpath to be used to issue an I/O. This is updated
  each time an I/O is issued or when the lunpath goes offline.

transient_q
  Pointer to the queue of bps (I/Os) when the LUN is in a transient
  state.

special_q
  Pointer to the queue of bps (I/Os) issued on this LUN either with the
  SIOC_IO or SIOC_IO_EXT ioctl, or through the cmdx interface.

wwid
  Specifies the actual LUN world wide identifier from EVPD page 0x83.

wwid_code_set
  Specifies the EVPD inq pg0x83 descriptor code set.

wwid_assoc
  Specifies the EVPD inq pg0x83 descriptor association.

wwid_type
  Specifies the EVPD inq pg0x80 descriptor type.

serial_len
  Specifies the EVPD inq pg0x80 descriptor size.

serial
  Specifies the EVPD inq pg0x80 descriptor.

open_cnt
  Specifies the current number of opens on the LUN, including block
  and character, and pass-through and class driver opens.

raw_open_cnt
  Specifies the current number of raw opens on the LUN, including
  pass-thru and class driver opens.

esctl_open_cnt
  Specifies the number of current pass-through opens.

total_path_cnt
  Specifies the current number of lunpaths to the LUN.

act_path_cnt
  Specifies the current number of active lunpaths to the LUN.

standby_path_cnt
  Specifies the current number of standby lunpaths to the LUN.

failed_path_cnt
  Specifies the current number of failed lunpaths to the LUN.

max_io_size
  Specifies the maximum I/O size (in bytes) to this LUN.

pref_io_size
  Specifies the preferred I/O size (in bytes).

io_id
  Specifies the unique I/O ID assigned to the LUN. This is a rolling
  counter.

active_ios
  Specifies the currently outstanding I/Os on this LUN. This value is
  calculated by the LUN support routine when the value is requested
  by totaling the number of individual lpt->active_ios.

lid
  Specifies the LUN index (dev_t minor) for this LUN.

dev
  Specifies the dev_t on which the first LUN open was performed.

dev_type
  Specifies the device type, either character (D_CHR) or block (D_BLK).
load_bal_policy  Specifies the I/O load balancing policy for this LUN. This attribute can be set and saved for disks, where is can be round robin or least command load policies.

transient_secs  Specifies the threshold time period after the LUN transitions out of the SLS_ONLINE state. During this time, the LUN remains in the transient condition and continues to queue received I/Os. When the time period elapses, the LUN is in a fatal condition; all received I/Os rejected with error. Drivers can set and save this value.

lbolt_online_exit  Specifies the lbolt value when the LUN manager transitioned out of the SLS_ONLINE_STATE. This is used to determine if a LUN is in a transient or fatal condition.

disable_flags  Specifies common DDR disable bits. The lower 32 bits are reserved for SCSI Services. The upper 32 bits are available for class drivers. This field is a shadow of the corresponding DDR attribute. The value is zero (0) if the DDR attribute does not exist for the LUN.

suspend_rsn  Specifies the reasons why this LUN is suspended. This is an exclusive OR of bit values from a set of reason bits.

suspend_pending  Specifies the pending suspend reasons to be processed.

prev_state  Specifies the previous saved state of the LUN.

ddsw  Pointer to the ddsw table of the class driver bound to this LUN. The default is the pass-through driver.

dd_lun  Specifies the class driver-specific LUN structure.

ki_handle  Specifies kernel instrumentation information.

sense_bytes  Specifies the number of bytes of sense data last received from the target.

sense_data  Specifies the sense data last received from the target.

num_trc_recs  Specifies the number of trace records.

trc_info  Specifies a trace buffer.

leg_mpath_enable  Specifies a tunable parameter that enables or disables multipathing when a legacy DSF is used. The default is ESCSI_TRUE.

io_start_timer  Specifies the LUN I/O start timer that is used when there are more then ESCSI_LUN_MAX_IO_START I/Os to be started on the LUN.

stat  Common LUN statistics.

SEE ALSO

escsi_ddsw_t(9S)
ESCSI_LUN_UNLOCK(9G)

NAME
ESCSI_LUN_UNLOCK - Macro to release the LUN lock.

SYNOPSIS
ESCSI_LUN_UNLOCK(__lun)

PARAMETERS
__lun Pointer to an escsi_lun_t structure.

DESCRIPTION
The ESCSI_LUN_UNLOCK macro releases the LUN lock. If the semaphore is not acquired, the
macro waits until the semaphore is available.
Class drivers must use the LUN lock to protect the integrity of the escsi_lun_t structure.

RETURN VALUES
None

CONSTRAINTS
The SCSI lock order of acquisition is:
1. LUN lock
2. lunpath

EXAMPLE
See the ESCSI_LUN_LOCK(9G) example.

SEE ALSO
ESCSI_LPT_LOCK(9G), ESCSI_LUN_LOCK(9G)
ESCSI_LUN_VSEMA(9G)

NAME

ESCSI_LUN_VSEMA - Macro to release the LUN semaphore.

SYNOPSIS

ESCSI_LUN_VSEMA(__lun)

PARAMETERS

__lun Pointer to an escsi_lun_t structure.

DESCRIPTION

The ESCSI_LUN_VSEMA macro releases the LUN semaphore.

Class drivers must use the LUN semaphore to protect the integrity of the lun->path_q in the context of the d_open and d_close entry points.

RETURN VALUES

None

CONSTRAINTS

The SCSI semaphore order of acquisition is:
1. Class driver LUN semaphore
2. LUN semaphore

EXAMPLE

See the ESCSI_LUN_PSEMA(9G) example.

SEE ALSO

ESCSI_LUN_DD_PSEMA(9G), ESCSI_LUN_PSEMA(9G)
ESCSI_MOVE_LIST(9G)

NAME
ESCSI_MOVE_LIST - Macro to move the list elements in one list to another list

SYNOPSIS
ESCSI_MOVE_LIST( escsi_list_t* __from_hdr),
                escsi_list_t* __to_hdr)

PARAMETERS
__from_hdr     Pointer to list header from which elements must be moved.
__to_hdr       Pointer to list header to which elements must be moved.

DESCRIPTION
The ESCSI_MOVE_LIST macro moves the list pointed to by header __from_hdr to __to_hdr. The
header __to_hdr is assumed to be empty; otherwise, the list that was being pointed to by __to_hdr
is lost.

RETURN VALUE
None

EXAMPLE
escsi_list_t from_hdr;
escsi_list_t to_hdr;
ESCSI_MOVE_LIST(&from_hdr, &to_hdr);

SEE ALSO

1028 SCSI Reference Pages
escsi_name_t(9S)

NAME

escsi_name_t - SCSI component name data structure (for example, target port name, node name)

SYNOPSIS

#include <sys/escsi_services.h>

typedef struct iovec escsi_name_t;

DESCRIPTION

This structure is used to store SCSI component names, such as target port name or node name. The component name is typically considered as array of bytes. The structure stores a pointer to the array and its length in bytes.

STRUCTURE MEMBERS

struct iovec {
#ifdef _KERNEL
        caddr_t  iov_base;
#else
        void     *iov_base;
#endif
        size_t   iov_len;
};

iov_base   Pointer to the memory address containing the component name.
iov_len    Specifies the length of the component, in bytes

CONSTRAINTS

None

SEE ALSO

escsi_iftgt_attr_t(9S)
escsi_new_lpt(9F)

NAME
escsi_new_lpt - Bind a new lunpath to the lun.

SYNOPSIS

void escsi_new_lpt(
    escsi_lun_t *lun,
    escsi_lpt_t *lpt
);

PARAMETERS

lun Pointer to a escsi_lun structure.
lpt Pointer to escsi_lpt structure.

DESCRIPTION

This is a default SCSI Services routine to bind a new lunpath to the LUN.
A class driver that provides a dd_lpt_init must call escsi_new_lpt to allow the SCSI
Services to do the common work involved in binding a new lunpath to a LUN. This service adds
the new lunpath to the escsi_lun_t->path_q queue and it increments some LUN statistics.
This routine must be called by the class drivers as the first step in its dd_lpt_init, after any
class driver specific synchronization or serialization steps have been performed.

RETURN VALUE

None

CONSTRAINTS

The caller must hold a LUN lock.

EXAMPLE

See the dd_lpt_init(9E) example.

SEE ALSO

dd_lpt_init(9E)
ESCSI_NEXT(9G)

NAME

ESCSI_NEXT - Macro to get list element next to the given list element/header.

SYNOPSIS

ESCSI_NEXT( escsi_list_t *__elem )

PARAMETERS

__elem    Pointer to list element or list header.

DESCRIPTION

The ESCSI_NEXT macro returns the next element pointed by list element pointer __elem.

RETURN VALUE

Pointer to the next escsi_list_t element.

CONSTRAINTS

None

EXAMPLE

escsi_lpt_t  *lpt;
escsi_list_t *next_elm;

ESCSI_LUN_LOCK(lun);
next_elm = ESCSI_NEXT(&lun->path_q);
while (next_elm != &lun->path_q) {
    lpt = ESCSI_GET_PARENT(next_elm);
    next_elm = ESCSI_NEXT(next_elm);

    /* Manipulate lunpath */
}

ESCSI_LUN_UNLOCK(lun);

SEE ALSO

ESCSI_GET_PARENT(9G), ESCSI_LUN_LOCK(9G)
escsi_node_to_addr(9F)

NAME

escsi_node_to_addr - Get the escsi_addr_t data structure of a SCSI object and the type of the object.

SYNOPSIS

#include <sys/escsi_services.h>

int escsi_node_to_addr(
    void *node,
    escsi_addr_t *addr,
    escsi_type_t *type
);

PARAMETERS

node  The I/O node handle of the SCSI object.
addr  Pointer to the escsi_addr_t data structure where the address information of the object is returned.

DESCRIPTION

This function returns address information of the SCSI object corresponding to an I/O node token and the type of the object. The following table shows the object type and the component of the address returned:

Table 5-4 Object, Type, and Address Components

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
<th>Address Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>lunpath</td>
<td>ESCSI_LPT</td>
<td>c, t, l</td>
</tr>
<tr>
<td>Target path</td>
<td>ESCSI_TP</td>
<td>c, t</td>
</tr>
<tr>
<td>Controller</td>
<td>ESCSI_CTLR</td>
<td>c</td>
</tr>
</tbody>
</table>

CALLING CONTEXT

The following are the contexts and usage of the above services:

• Interface drivers. CLIs accept class and instance numbers as input parameters to identify controller or target objects. To get the interface driver’s corresponding object handles, do the following:
  1. Call class_get_node to get the I/O node.
  2. Call escsi_node_to_addr to get escsi_addr_t of the corresponding SCSI object.
  3. Call escsi_ifctrlr_get or escsi_iftgt_get to get the interface driver’s object handle.

• To access I/O node token of any of the SCSI objects, interface drivers can invoke escsi_addr_to_node.

RETURN VALUE

EINVAL  SCSI Services does not know this node, or the object does not has an escsi_addr_t data structure. Typically, a LUN does not have an escsi_addr_t data structure associated with it.

ESUCCESS The addr and object type returned are valid.
CONSTRAINTS
This is a trusted interface. Drivers must pass the I/O node only if it is known to the SCSI Services. The caller must make sure that the I/O node in question is valid across the call. Caller is expected to provide memory to accommodate SCSI address in $addr$. The above services can be used by interface drivers to get their controller or target object's handles, or to get the I/O node given $escsi_{addr}\_t$.

EXAMPLES
See the $escsi\_ifctrl\_get(9F)$, $escsi\_iftp\_get(9F)$, and $escsi\_iflpt\_get(9F)$ examples.

SEE ALSO
$escsi\_addr\_t(9S)$, $escsi\_type\_t(9S)$
ESCSI_OWNS_IOBJ_LOCK(9G)

NAME
ESCSI_OWNS_IOBJ_LOCK - Macro to check if I/O object lock is currently owned

SYNOPSIS
ESCSI_OWNS_IOBJ_LOCK ( escsi_iobj_t *__iobj)

PARAMETERS
__iobj Pointer to an I/O object structure.

DESCRIPTION
The ESCSI_OWNS_IOBJ_LOCK macro checks whether the I/O object spinlock is currently owned by the caller.

RETURN VALUES
0 The I/O object lock is not owned.
1 The I/O object lock is owned.

CONSTRAINTS
None

EXAMPLE
if (ESCSI_OWNS_IOBJ_LOCK(iobj) == FALSE)
    ESCSI_LOCK_IOBJ(iobj);

SEE ALSO
ESCSI_IOBJ_LOCK(9G), ESCSI_IOBJ_UNLOCK(9G)
ESCSI_OWNS_LEG_LUN_LOCK(9G)

NAME

ESCSI_OWNS_LEG_LUN_LOCK - Macro to test whether a legacy LUN lock is held.

SYNOPSIS

ESCSI_OWNS_LEG_LUN_LOCK(__leg_lun)

PARAMETERS

__leg_lun Pointer to an escsi_leg_lun_t structure.

DESCRIPTION

The ESCSI_OWNS_LEG_LUN_LOCK macro tests whether a legacy LUN lock is held.

RETURN VALUES

ESCSI_TRUE The legacy LUN lock is held.
ESCSI_FALSE The legacy LUN lock is not held.

CONSTRAINTS

None

EXAMPLE

escsi_leg_lun_t *leg_lun;
ESCSI_ASSERT(ESCSI_OWNS_LEG_LUN_LOCK(leg_lun));

SEE ALSO

ESCSI_LEG_LUN_LOCK(9G)
ESCSI_OWNS_LPT_LOCK(9G)

NAME

ESCSI_OWNS_LPT_LOCK - Macro to test whether lunpath lock is held.

SYNOPSIS

ESCSI_OWNS_LPT_LOCK(__lpt)

PARAMETERS

__lpt Pointer to an escsi_lpt_t structure.

DESCRIPTION

The ESCSI_OWNS_LPT_LOCK macro tests whether a lunpath lock is held.

RETURN VALUES

ESCSI_TRUE The lunpath lock is held.
ESCSI_FALSE The lunpath lock is not held.

CONSTRAINTS

None

EXAMPLE

escsi_lpt_t *lpt;
ESCSI_ASSERT(ESCSI_OWNS_LPT_LOCK(lpt));

SEE ALSO

ESCSI_LPT_LOCK(9G)
ESCSI_OWNS_LUN_LOCK(9G)

NAME
ESCSI_OWNS_LUN_LOCK - Macro to test whether a LUN lock is held.

SYNOPSIS
ESCSI_OWNS_LUN_LOCK(__lun)

PARAMETERS
__lun    Pointer to an escsi_lun_t structure.

DESCRIPTION
The ESCSI_OWNS_LUN_LOCK macro tests whether a LUN lock is held.

RETURN VALUES
ESCSI_TRUE    The LUN lock is held.
ESCSI_FALSE   The LUN lock is not held.

CONSTRAINTS
None

EXAMPLE
esxi_lun_t *lun;
ESCSI_ASSERT(ESCSI_OWNS_LUN_LOCK(lun));

SEE ALSO
ESCSI_LUN_LOCK(9G)
escsi_owns_sema(9F)

NAME
escsi_owns_sema - Determine whether a semaphore is already held.

SYNOPSIS

escsi_bool_t escsi_owns_sema(
    escsi_bsema_t *sema
);

PARAMETERS

sema Pointer to an eSCSI semaphore.

DESCRIPTION
The escsi_owns_sema routine determines whether a semaphore is already held.

RETURN VALUES

ESCSI_TRUE The semaphore is held.
ESCSI_FALSE The semaphore is not held.

CONSTRAINTS

None

SEE ALSO

escsi_init_sema(9F), escsi_psema(9F), escsi_vsema(9F)
ESCSI_PREPEND_LIST(9G)

NAME

ESCSI_PREPEND_LIST - Macro to insert one list in front of another list.

SYNOPSIS

```
ESCSI_PREPEND_LIST (escsi_list_t *__hdr1,
                    escsi_list_t *__hdr2)
```

PARAMETERS

__hdr1  Pointer to a list header.
__hdr2  Pointer to a list header.

DESCRIPTION

The `ESCSI_PREPEND_LIST` macro inserts the list pointed to by __hdr2 at the head of the list pointed to by __hdr1. The new list to be pointed to by the __hdr1. The __hdr2 argument must not be an empty list during the call.

RETURN VALUE

None

CONSTRAINTS

None

EXAMPLE

```
escsi_list_t list1_hdr;
escsi_list_t list2_hdr;
ESCSI_PREPEND_LIST(&list1_hdr, &list2_hdr);
```

SEE ALSO

`ESCSI_APPEND_LIST(9G)`
ESCSI_PREV(9G)

NAME
ESCSI_PREV -- Macro to get list element previous to the given list element/header

SYNOPSIS
ESCSI_PREV(   escsi_list_t *__elem )

PARAMETERS
__elem     Pointer to a list element or list header.

DESCRIPTION
The ESCSI_PREV macro returns the previous element pointed to by the __elem list element pointer.

RETURN VALUE
Pointer to previous escsi_list_t element.

CONSTRAINTS
None

EXAMPLE
prev = ESCSI_PREV(&activeq_hdr);

SEE ALSO
ESCSI_NEXT(9G)
escsi_process_sense(9F)

NAME

escsi_process_sense - Default sense action routine for processing I/O sense data.

SYNOPSIS

int escsi_process_sense(
    struct buf *bp,
    escsi_sense_action_t *sense_list,
    size_t sense_cnt
);

PARAMETERS

bp Pointer to a buf structure.
sense_list Pointer to a sense list structure containing sense data.
sense_cnt Number of elements in the sense list.

DESCRIPTION

The escsi_process_sense routine is the default for processing I/O sense data. Class drivers can use this function by default to handle check conditions and process sense data. A driver-specific sense data processing routine must invoke escsi_process_sense before returning.

RETURN VALUES

-1 The I/O is to be completed back to the upper layers.
>=0 The I/O is to be retried. Delay a specific interval (in milliseconds) before retrying.

CONSTRAINTS

Do not hold nay locks when calling this routine.
This function does not sleep.

EXAMPLE

Following is an example where escsi_process_sense is used by default to process a CHECK condition:

escsi_sense_action_t xxx_sense_list[] = {
    ...
}

escsi_status_action_t xxx_status_list[] = {
    ...
    {S_CHECK_CONDITION, (escsi_action_t)escsi_process_sense,
     (intptr_t)&xxx_sense_list[0],
     sizeof(xxx_sense_list)/sizeof(xxx_sense_list[0]),
     0, 0
    }
    ...
}

Following is an example where a driver-specific routine handles sense data:

escsi_sense_action_t xxx_sense_list[] = {
    ...
}

escsi_status_action_t xxx_status_list[] = {
    ...
}
{S_GOOD, escsi_action, 0, 0, EEI_NONE, 0},
{S_CHECK_CONDITION, xxx_sense_data, 0, 0, EEI_NONE, 0},
...
}

int
xxx_sense_data(struct buf *bp, intptr_t flags, intptr_t errno,
intptr_t eei, intptr_t msecs)
{
    escsi_scb_t *scb = (escsi_scb_t *)bp->B_SCB;
    escsi_lpt_t *lpt = (escsi_lpt_t *)bp->B_LPT;
    escsi_lun_t *lun = lpt->lun;
    sense_data_t *sense = (sense_data_t*) scb->sense_data;
    uint32_t sense_info = 0;

    if(scb->sense_data == NULL) {
        /* No sense data available */
        return escsi_action(bp, flags, (intptr_t)EIO,
                            (intptr_t)EEI_NONE, (intptr_t)0);
    }
    /* Process sense data */
    .
    .
    /* Must invoke escsi_process_sense() before returning */
    return escsi_process_sense(bp, xxx_sense_list,
                                sizeof(xxx_sense_list)/sizeof(escsi_sense_action_t));
}

SEE ALSO

escsi_action(9F)
escsi_psema(9F)

NAME

escsi_psema - Acquire an eSCSI semaphore.

SYNOPSIS

void escsi_psema(
    lock_t *lock,
    escsi_bsema_t *sema
);

PARAMETERS

lock Pointer to the lock used to protect the escsi_bsema_t semaphore.
sema Pointer to the escsi_bsema_t semaphore.

DESCRIPTION

The escsi_psema routine acquires an eSCSI semaphore. If the semaphore is available, this
routine starts holding it. Otherwise, the routine sleeps until the semaphore is released. The lock
parameter is mandatory.

RETURN VALUES

None

CONSTRAINTS

The lock parameter is mandatory.
Must be called in a thread context because the routine can sleep.

EXAMPLE

None

SEE ALSO

escsi_init_sema(9F), escsi_owns_sema(9F), escsi_vsema(9F)
NAME
escsi_reg_leg_bus - Legacy (virtual) bus hardware path registration with SCSI Services

SYNOPSIS
int escsi_reg_leg_bus(
    uint32_t bus_inst
);

PARAMETERS
bus_inst Specifies the instance number of the legacy bus in the ext_bus driver class.

DESCRIPTION
This function is invoked by the interface driver to register a legacy bus with the SCSI Services. If the legacy bus does not map to a new controller (for example, FiberChannel or iSCSI legacy virtual bus), this function is invoked in the context of the ifsw->if_leg_hwp_build entry point after the interface driver has registered the virtual bus with the I/O infrastructure by invoking wsio_reg_legacy. This routine registers a legacy virtual bus with the SCSI Services using the bus instance returned by wsio_reg_legacy as bus_inst parameter.

If the legacy bus maps to a new controller (for example, pSCSI ext_bus controller), this service is invoked in the context of the driver attach entry point, after the controller is registered with the SCSI Services using escsi_ctlr_reg. The bus_inst parameter is the ctrlr isc instance number.

If the legacy bus has previously been registered, the routine returns ESUCCESS.

RETURN VALUE
ESUCCESS The registration is successful, the legacy bus already exists.
EINVAL Called in a non-blocking context or with spinlock held.
EOVERFLOW More than 256 ext_bus driver instances have been registered. The legacy model only supports a maximum of 256 (0 to 255) ext_bus driver instances.

CONSTRAINTS
This function must be called in a blocking context.
No spinlocks must be held by the caller.

EXAMPLE
See if_leg_hwp_build example

SEE ALSO
if_leg_hwp_build(9F), escsi_reg_leg_tgt(9F), escsi_reg_leg_lun(9F)
escsi_reg_leg_lun(9F)

NAME

escsi_reg_leg_lun - Register legacy (virtual) LUN path hardware path registration with SCSI Services and with the I/O infrastructure

SYNOPSIS

int escsi_reg_leg_lun(
    escsi_reg_leg_lun_t *leg_lun_reg
);

PARAMETERS

leg_lun_reg Pointer to the escsi_reg_leg_lun_t structure initialized by the interface driver.

DESCRIPTION

This function is invoked by the interface driver in the context of the ifsw->if_leg_hwp_build entry point. It registers a legacy (virtual) lunpath with the SCSI Services and with I/O infrastructure. It follows the invocation of the escsi_reg_leg_tgt service for the registration of the parent legacy target hardware path.

This function must be invoked even though the legacy LUN hardware path has already been registered in a previous instance of scanning.

The escsi_reg_leg_lun_t input structure has the following fields:

addr The escsi_addr_t corresponding to the new lunpath to which the legacy lunpath maps. This allows the I/O infrastructure to map the legacy lunpath I/O node to the right lunpath I/O node.

bus_inst The bus instance number that was passed in the previous invocation of escsi_reg_leg_tgt for the registration of the parent legacy target hardware path.

tgt_id The legacy (virtual) SCSI-2 target ID.

lun_id The legacy (virtual) SCSI-2 LUN ID.

lun_hw_path The absolute legacy lunpath hardware path.

RETURN VALUE

ESUCCESS If registration of a legacy LUN is successful.

EINVAL The registration of the legacy LUN with the I/O infrastructure failed; invalid parameters were passed in (for example, invalid bus_inst or invalid tgt_id); or the function was invoked with lock held or in ICS context.

CONSTRAINTS

This function must be called in blocking context.

No spinlocks must be held by the caller.

EXAMPLE

See if_leg_hwp_build example.

SEE ALSO

if_leg_hwp_build(9F), escsi_reg_leg_bus(9F), escsi_reg_leg_lun_t(9S), escsi_reg_leg_tgt(9F)
NAME

escsi_reg_leg_tgt - Legacy (virtual) target path hardware path registration with SCSI Services and I/O infrastructure

SYNOPSIS

int escsi_reg_leg_tgt(
    escsi_reg_leg_tgt_t *reg_leg_tgt
);

PARAMETERS

reg_leg_tgt Pointer to the escsi_reg_leg_tgt_t structure initialized by the interface driver.

DESCRIPTION

This function is invoked by the interface driver in the context of the ifsw->if_leg_hwp_build entry point. It registers a legacy target path with the SCSI Services and with the I/O infrastructure. This function must be invoked even though the legacy target path hardware path has already been registered in a previous instance of scanning. The escsi_reg_leg_tgt_t input structure contains the following fields:

node Specifies the component I/O node to which the legacy target path maps. This enables the I/O infrastructure to map the legacy target path I/O node to the right target path I/O node.

bus_inst Specifies the bus instance number that was passed or returned by the previous invocation of escsi_reg_leg_bus for the registration of the parent legacy bus hardware path.

tgt_id Specifies the legacy virtual SCSI-2 target ID.

tgt_hw_path Specifies the absolute legacy target path hardware path.

RETURN VALUE

ESUCCESS The registration of the legacy target is successful.

EINVAL The registration of the legacy target with the I/O infrastructure failed; invalid parameters were passed in (for example, invalid bus_inst); or called in a non-blocking context or with spinlock held.

EXAMPLE

See if_leg_hwp_build example.

CONSTRAINTS

This function must be called in blocking context. No spinlocks must be held by the caller.

SEE ALSO

if_leg_hwp_build(9F), excsi_ref_leg_bus(9F), escsi_reg_leg_lun(9F), escsi_reg_leg_tgt_t(9S)
escsi_reg_leg_tgt_t(9S)

NAME

escsi_reg_leg_tgt_t - Structure passed as argument of escsi_reg_leg_tgt to register a legacy virtual target with the I/O infrastructure and the SCSI Services

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

Describes the legacy target path structure. The interface driver initializes this structure and passes it in the escsi_reg_leg_tgt KPI to SCSI Services when registering legacy target path hardware paths (escsi_ifsw_t->if_leg_hwp_build entry point).

STRUCTURE MEMBERS

typedef struct {
    void            *node;
    uint8_t         bus_inst;
    uint8_t         tgt_id;
    hw_path_t       *tgt_hw_path;
} escsi_reg_leg_tgt_t;

node        Specifies the new style target path I/O node with which the legacy target is associated.

bus_inst    Specifies the legacy virtual bus instance number.

tgt_id      Specifies the legacy virtual target ID.

tgt_hw_path Specifies the absolute hardware path needed to register the legacy hardware path of the virtual target with WSIO.

CONSTRAINTS

None

SEE ALSO

escsi_reg_leg_tgt(9F), hw_path_t(9S)
NAME
ESCSI_RELE - Macro to release a SCSI object so it can be deleted.

SYNOPSIS
ESCSI_RELE(__obj, __own_lock)

PARAMETERS
__obj Pointer to an escsi_info_t structure of the SCSI object.
__own_lock Boolean to determine whether the macro has to hold and release the SCSI object
lock while performing the release operation.

DESCRIPTION
The ESCSI_RELE macro releases a SCSI object so it can be deleted. After invoking ESCSI_HOLD,
when an operation no longer needs to protect a SCSI object from deletion, it invokes ESCSI_RELE.
The SCSI object can only be deleted when the last ESCSI_RELE is invoked.

RETURN VALUES
None

CONSTRAINTS
ESCSI_HOLD must be called before ESCSI_RELE.
If the SCSI object lock is not already held while calling ESCSI_RELE, the caller must set the
__owns_lock parameter to ESCSI_FALSE.

EXAMPLE
See the ESCSI_HOLD(9G) example.

SEE ALSO
ESCSI_HOLD(9G), escsi_lookup_obj(9F), SIOC_ATTR(9G)
escsi_scb_flags_t(9S)

NAME
escsi_scb_flags_t - Flags associated with the escsi_sbc_t structure

SYNOPSIS
#include <io/escsi_services.h>

DESCRIPTION
This structure defines a set of flags that can be used in the escsi_scb_t structure.

STRUCTURE MEMBERS
typedef enum {
    SCBF_UNTAGGED    = 0x001,    /* set by SCSI Services to
        * indicate that the device
        * is untagged based on
        * inquiry data.
        */
    SCBF_WDTR        = 0x002,    /* The following flags:
        * SCBF_WDTR, SCBF_SDTR
        * and SCBF_NO_DISC are
        * are meant to allow
        * support for the legacy
        * SCB_WDTR, SCB_SDTR
        * and SCB_NO_DISC flag
        * definitions.
        */
    SCBF_SDTR        = 0x004,
    SCBF_NO_DISC     = 0x008,
    SCBF_SIM_TAG     = 0x010,    /* set by SCSI Services to
        * indicate device supports
        * tagged queuing. Simple
        * Queue tag.
        */
    SCBF_ORD_TAG     = 0x020,    /* ORDERED TAG and HEAD OF Q
        * tag are reserved for future
        * use. Currently not used
        * by SCSI Services.
        */
    SCBF_HOQ_TAG     = 0x040,    /* see comment above */
    SCBF_HIPRI       = 0x080,    /* set by SCSI Services to
        * indicate a high priority
        * IO. A high priority IO
        * is not subject to the
        * flow control on the
        * lunpath.
        */
    SCBF_LOGGED      = 0x100,    /* This is an internal flag
        * used by SCSI Services to
        * prevent excessive logging.
        */
    SCBF_NO_LOG      = 0x200     /* This is an internal flag
        * that can be set by the
        * passthru driver or by
        * i/f driver to disable
        * logging on a per scb basis.
        * Note that the class driver
        * can override this setting
        * by setting the log always
        * flag in its status handling
        * list.
        */
    SCBF_PMUTE    = 0x400        /* Power Mute Suppress Flag.
        * When set, the command
        * and response sequences
        * will be handled in a
        * manner different than the
        * default settings.
        */
    SCBF_PMEH     = 0x800        /* Power Manage Extension Flag.
        * When set, the command
        * and response sequences
        * will be handled in a
        * manner different than the
        * default settings.
        */
}
} escsi_scb_flags_t;

CONSTRAINTS
None

SEE ALSO

escsi_scb_t(9S)
escsi_scb_t(9S)

NAME
escsi_scb_t - scb structure definition shared with interface drivers

SYNOPSIS
#include <io/escsi_services.h>

DESCRIPTION
The escsi_scb_t structure holds the I/O request SCSI specific details (for example, cdb) and
SCSI Services management information (for example, max_msecs). It also holds a reference to
a buf structure. This reference persists until the I/O is returned back to the upper layers.

STRUCTURE MEMBERS
typedef struct {
    escsi_list_t   list;
    int         version;
    escsi_iobj_t   *iobj;
    void         *if_tp;
    void         *if_lpt;
    void         *if_scb;
    void         *ulp_scb;
    struct buf      *bp;
    uint64_t      lun_id
    escsi_scb_flags_t   flags;
    uint8_t      *cdb;
    uint32_t      cdb_len;
    uint32_t      max_msecs;
    uint32_t      data_resid;
    uint32_t      cdb_status;
    uint32_t      sense_status;
    uint32_t      sense_bytes;
    uint8_t      *sense_data;
    escsi_iodone_t   cbfn
    intptr_t      *ki_handle;
} escsi_scb_t;

list    Specifies the linked list element used to chain scbs in the free pool.
version The Application Binary Interface (ABI) version field. This value is set by the
          scb allocation routine.
iobj    Back pointer to the shared I/O object structure. This value is set by the scb
          allocation routine.
if_tp   Back pointer to the interface driver private target path structure.
if_lpt  Back pointer to the interface driver private lunpath structure.
if_scb  Pointer to the interface driver private scb structure.
ulp_scb Pointer to the SCSI Services private scb structure.
bp      Specifies a back pointer to the structure.
lun_id  Specifies the LUN ID as reported in the REPORT LUNS command.
flags   An enum of type escsi_scb_flags_t. See escsi_scb_flags_t(9S) for more
        information.
cdb     Pointer to the SCSI Command Data Block (CDB).
cdb_len Specifies the SCSI CDB length.
max_msecs Specifies the timeout value (in milliseconds). A value of 0 means the interface
driver must not time the I/O.
data_resid  Specifies the number of bytes of data that were not transferred. This is the data residue of the operation.

cdb_status  Specifies the I/O status.

sense_status  Specifies the status of the sense operation.

sense_bytes  Specifies the number of bytes of sense data.

sense_data  Sense information.

cbfn  Specifies the SCSI callback function that the interface driver calls on I/O completion.

ki_handle  Specifies the handle to use to collect kmetrics on this I/O.

SEE ALSO

buf(9S), escsi_iobj_t(9S), escsi_iiodone_t(9S), escsi_list_t(9S), escsi_scb_flags_t(9S)
escsi_sctl_io_flags_t(9S)

NAME

escsi_sctl_io_flags_t - Flags that can be set in the esctl_io data structure.

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

These flags describe the nature of the I/O issued via the SIOC_IO_EXT ioctl. These flags can be issued from user space.

STRUCTURE MEMBERS

typedef enum {
    _ESCTL_READ          = 0x01,
    _ESCTL_INIT_WDTR     = 0x02,
    _ESCTL_INIT_SDTR     = 0x04,
    _ESCTL_NO_DISC       = 0x08,
    _ESCTL_IO_LPT        = 0x10,
    _ESCTL_IO_LOG        = 0x20,
    _ESCTL_FROM_KERNEL   = 0x800
} escsi_sctl_io_flags_t;

_ESCTL_READ Indicates an I/O read operation.

_ESCTL_INIT_WDTR Parallel SCSI: Wide data transfer. Maintained for backward compatibility with legacy applications that issue sending pass-through SCSI commands via SIOC_IO.

_ESCTL_INIT_SDTR Parallel SCSI: Synchronous transfer. Maintained for backward compatibility with legacy applications that issue sending pass-through SCSI commands via SIOC_IO.

_ESCTL_NO_DISC Parallel SCSI: No disconnection during transfer. Maintained for backward compatibility with legacy applications that issue sending pass-through SCSI commands via SIOC_IO.

_ESCTL_IO_LPT The I/O is to be issued only on the specified lunpath.

_ESCTL_IO_LOG Log the I/O in case of failure. By default, pass-through I/O is not logged.

_ESCTL_FROM_KERNEL The ioctl is being issued by the kernel. User applications must not set this flag. The numeric value of this flag must be the same as SCTL_FROM_KERNEL.

CONSTRAINTS

None

SEE ALSO

esctl_io_t(9S)
escsi_sense_action_t(9S)

NAME
escsi_sense_action_t - Sense list definition structure.

SYNOPSIS
#include <io/escsi_common.h>

DESCRIPTION
The sense list of a class driver processes the sense information (for example, sense key, ASC, and ASCQ) in scb->sense_data.

STRUCTURE MEMBERS
typedef struct {
    int status;
    int16_t error_code;
    int16_t sense_key;
    int16_t asc;
    int16_t ascq;
    escsi_action_t action;
    intptr_t flags;
    intptr_t errno;
    intptr_t eei;
    intptr_t msecs;
    intptr_t rsvd[5];
} escsi_sense_action_t;

status Specifies the status of the required sense operation.
error_code Specifies the error code indicating current or deferred error.
sense_key
asc
ascq
action Specifies the action routine to process the sense data.
flags Specifies whether the default action for the given sense information is to retry or not.
errno Specifies the errno to be returned in bp->b_errno.
eei Specifies the eei to be returned in bp->b_eei.
msecs Specifies the time (in milliseconds) to wait before retrying the I/O operation.
rsvd Reserved for internal use. Do not modify.

CONSTRAINTS
None

SEE ALSO

1054 SCSI Reference Pages
ESCSI_SET_ADDR(9G)

NAME

ESCSI_SET_ADDR - Macro to set components of the escsi_addr_t data structure.

SYNOPSIS

#include <sys/escsi_services.h>

ESCSI_SET_ADDR(__addr, __c, __t, __l);

PARAMETERS

__addr Pointer to a escsi_addr_t data structure whose component must be set.
__c Controller instance number.
__t Target path instance number.
__l LUN identifier.

DESCRIPTION

The ESCSI_SET_ADDR macro initializes the component of a escsi_addr_t data structure.

RETURN VALUE

None

EXAMPLE

ESCSI_SET_ADDR(&activeq_hdr, lpt);
}

CONSTRAINTS

None

SEE ALSO

escsi_addr_t(9S), ESCSI_SET_CBFN(9G)
ESCSI_SET_CBFN(9G)

NAME

ESCSI_SET_CBFN - Macro to set components of an escsi_cbfn_t data structure.

SYNOPSIS

#include <sys/escsi_services.h>

ESCSI_SET_CBFN(__cbfn, __cbfunc, __cbarg);

PARAMETERS

__cbfn Pointer to an escsi_cbfn_t data structure whose components must be set.
__cbfunc Pointer to the callback function.
__cbarg Argument to pass to the callback function.

DESCRIPTION

The ESCSI_SET_CBFN macro sets values for individual members of a escsi_cbfn_t data structure.

RETURN VALUE

None

EXAMPLE

ESCSI_SET_CBFN(&activeq_hdr, lpt);
}

CONSTRAINTS

None

SEE ALSO

escsi_cbfn_t(9S), ESCSI_SET_ADDR(9G)
NAME
ESCSI_SET_PARENT - Macro to set the parent pointer of a linked list element

SYNOPSIS
ESCSI_SET_PARENT( escsi_list_t __elem,
    void *parent);  

PARAMETERS
__elem Pointer to a list element.
__parent Pointer to a parent structure.

DESCRIPTION
The ESCSI_SET_PARENT macro sets the parent pointer of a list element.

RETURN VALUE
None

EXAMPLE
escsi_lpt_t *lpt;

    /*
     * Set lpt as the parent pointer of the list
     * element lpt->path_qelm
     */
    ESCSI_SET_PARENT(&lpt->path_qelm, lpt);

CONSTRAINTS
None

SEE ALSO
ESCSI_GET_PARENT(9G)
NAME
escsi_status_action_t - Status list definition structure.

SYNOPSIS
#include <io/escsi_common.h>

DESCRIPTION
The status list of a class driver processes the scb->cdb_status.

STRUCTURE MEMBERS
typedef struct {
    int               status;
    escsi_action_t    action;
    intptr_t          flags;
    intptr_t          errno;
    intptr_t          eei;
    intptr_t          msecs;
    intptr_t          rsvd[5];
} escsi_status_action_t;

status  Specifies the cdb status value.
action   Specifies the action routine to handle the cdb status value.
flags    Specifies flags to indicate whether to retry or not retry the default action for the given status.
errno    Specifies the errno to be returned in bp->b_errno.
eei      Specifies the eei to be returned in bp->b_eei.
msecs    Specifies the amount of time (in milliseconds) to wait before retrying the I/O operation.
rsvd     Reserved.

CONSTRAINTS
None

SEE ALSO
escsi_strategy(9F)

NAME

escsi_strategy - Process I/Os issued on a LUN.

SYNOPSIS

int escsi_strategy(
    struct buf *bp
);

PARAMETERS

bp  Pointer to buf structure.

DESCRIPTION

The escsi_strategy routine is a generic SCSI Services interface that processes I/O requests issued on a LUN DSF. In the process, it calls various class driver escsi_ddsw_t entry point routines for any class driver specific operations.

Drivers issuing block I/O requests on a new LUN DSF use escsi_strategy as the driver drv_ops_t->d_strategy entry point for the HP-UX disk driver. Drivers issuing raw I/O requests on a LUN DSF using the drv_ops_t->d_read and drv_ops_t->d_write entry points can also invoke escsi_strategy with the physio routine.

The escsi_strategy routine processes an I/O request as follows:

1. Calls the escsi_ddsw_t->dd_strategy entry point, which is expected to return 0 if the I/O request has been successfully processed and queued for further processing. A non-zero return indicates class driver failed the I/O.
2. Starts the I/O request queued on the LUN by the class driver by calling the escsi_ddsw_t->dd_start entry point.
3. Selects the lunpath from the active path queue on which the I/O is issued.
4. Calls escsi_ddsw_t->dd_io_init for the class driver to initialize the driver specific fields of the escsi_scb_t structure.
5. Issues the I/O to the interface driver corresponding to the lunpath that was selected.
6. Returns the I/O request completion status to the upper layers.

RETURN VALUE

0  Always returns 0.

CONSTRAINTS

Do not hold any locks when calling this routine.

EXAMPLE

static drv_ops_t xxx_ops = {
    xxx_open,   /* driver d_open entry point */
    xxx_close,  /* driver d_close entry point */
    escsi_strategy,  /* driver d_strategy entry point */
    NULL,   /* driver d_dump entry point */
    NULL,   /* driver d_psize entry point */
    NULL,   /* Reserved */
    xxx_read,   /* driver d_read entry point */
    xxx_write,  /* driver d_write entry point */
    NULL,   /* driver d_ioctl entry point */
    NULL,   /* driver d_select entry point */
    NULL,   /* driver d_option1 entry point */
    NULL,   /* driver driver pfilter entry point */
    NULL,   /* driver d_psize1 entry point */
};
NULL, /* driver callback */
NULL, /* driver d_aio_ops entry point */
C_ALLCLOSES | C_MGR_IS_MP | C_OPAQDEV /* d_flags */
};

xxx_read(dev_t dev, struct uio *uio)
{
  
  error = physio(escsi_strategy, (struct buf *) NULL, dev, B_READ,
                 xxx_minphys, uio);
  
  return (error);
}

xxx_write(dev_t dev, struct uio *uio)
{
  
  error = physio(escsi_strategy, (struct buf *) NULL, dev, B_WRITE,
                 xxx_minphys, uio);
  
  return (error);
}

SEE ALSO

dd_io_init(9E), dd_start(9E), dd_strategy(9E), escsi_leg_strategy(9E)
NAME
escsi_tag_alloc - Called by interface driver to allocate a tag from SCSI Services

SYNOPSIS

int64_t escsi_tag_alloc(
    escsi_scb_t *scb
);

PARAMETERS

scb  Pointer to a scb structure.

DESCRIPTION

This function is called by interface driver during I/O processing to allocate a tag from SCSI Services. The tag width is not be more than the if_tag_size specified by the interface driver in escsi_ifsw. Interface driver is responsible for synchronizing concurrent calls to the escsi_tag_alloc and escsi_tag_free routines by holding its own lock that is appropriate for the tag scope defined.

In order to use this service, the interface driver must specify the tag allocation scope using the if_flags field in escsi_ifsw_t. This field must be set to IF_IOBJ_TAGS, IF_TP_TAGS, or IF_CTLR_TAGS.

The interface driver must synchronize concurrent calls to the escsi_tag_alloc and escsi_tag_free routines. SCSI Services does not hold any spinlock to synchronize these calls.

RETURN VALUE

On successful completion (a tag is available), the routine returns a 64-bit tag. Otherwise, the routine returns -1.

CONSTRAINTS

This function can be called by interface driver with spinlocks held.

EXAMPLE

if_scb->tag = escsi_tag_alloc(scb);
if (if_scb->tag == -1) {
    /* No tags; keep I/O pending in internal queue */
    .
    .
    .
}

SEE ALSO

escsi_tag_free(9F)
escsi_tag_free(9F)

NAME

escsi_tag_free - Called by interface driver to release a tag allocated from SCSI Services

SYNOPSIS

void escsi_tag_free(
    escsi_scb_t *scb,
    int64_t tag
);

PARAMETERS

scb Pointer to a scb structure.
tag Tag value to be released.

DESCRIPTION

The escsi_tag_free function is used by interface driver during I/O completion processing to release a tag that was earlier allocated from SCSI Services. It is the responsibility of interface driver to synchronize concurrent calls to the escsi_tag_alloc and escsi_tag_free routines by holding its own lock that is appropriate for the tag scope defined.
The caller must ensure that the tag value passed to this function was earlier allocated using escsi_tag_alloc. Any violation of these rules can potentially lead to corruption of the tag pool.

RETURN VALUE

None

CONSTRAINTS

This function can be called by interface driver with spinlocks held.

EXAMPLE

escsi_tag_free(scb, if_scb->tag);

SEE ALSO

escsi_tag_alloc(9F)
NAME

escsi_task_t - Task management function definitions

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

This structure defines flags for the different task management operations supported by an interface driver.

STRUCTURE MEMBERS

typedef enum {
    ABORT_TASK_SET      = 0x01,
    CLEAR_TASK_SET      = 0x02,
    LUN_RESET           = 0x04,
    WARM_TARGET_RESET   = 0x08,
    COLD_TARGET_RESET   = 0x10,
    BUS_RESET           = 0x20
} escsi_task_t;

CONSTRAINTS

None

SEE ALSO

escsi_ifsw_t(9S), escsi_tm_arg_t(9S)
NAME
escsi_tgt_node_cb - Target path I/O node callback function.

SYNOPSIS
int escsi_tgt_node_cb(
    void *handle,
    io_events_t event,
    uintptr_t event_info
);

PARAMETERS
handle     Target path I/O node token.
event      Event passed in by I/O infrastructure.
event_info Event specific data passed in by I/O infrastructure.

DESCRIPTION
This function is called by interface driver to process target path I/O node callback CB_SCAN_ALL event.
As part of target path registration with the SCSI Services (escsi_tgt_reg), the interface driver would have registered its own target path I/O node callback function in escsi_iftgt_reg_t->cb_func and its argument in escsi_iftgt_reg_t->cb_arg. The interface driver target path I/O node callback routine must call the escsi_tgt_node_cb SCSI service whenever it is invoked by I/O infrastructure with the following event:
CB_SCAN_ALL Start of target path scan operation.

RETURN VALUE
GIO_SUCCESS Event is processed successfully.
GIO_ERROR Event is not processed.
GIO_E_FUNC_NOT_SUPPORTED Event not supported.

CONSTRAINTS
Do not hold any spinlocks while calling this function.

EXAMPLE
In this example, an interface driver target path I/O node callback routine invokes escsi_tgt_node_cb to handle the CB_SCAN_ALL event:
xxx_drv_tp_node_cb(void *handle, void *arg, io_events_t event,
    uintptr_t event_info)
{
    switch(event) {
        case CB_SCAN_ALL:
            escsi_tgt_node_cb(handle, event, event_info);
            break;
        .
        .
        .
    } /* switch */
    .
    .
} /* xxx_drv_tp_node */
SEE ALSO

wsio_node_claim(9F), escsi_ctlr_node_cb(9F)
NAME
escsi_tgt_reg - SCSI target registration interface

SYNOPSIS

```c
void escsi_tgt_reg(
    escsi_iftgt_reg_t *iftgt_reg
);
```

PARAMETERS

iftgt_reg Pointer to a target registration structure allocated by interface driver.

DESCRIPTION

This function is called by an interface driver to register a target with SCSI stack. This is usually called when the interface driver discovers a new target as part of its controller probe operation. Before calling this, interface driver is expected to allocate the `escsi_iftgt_reg_t`, `escsi_iftgt_attr_t`, and `escsi_iftgt_stat_t` data structures. This function is an asynchronous interface.

The interface driver needs to provide the hardware element information of the target path in `hwp` field of `escsi_iftgt_reg_t`. If the hardware element information of target path is more than 64 bits, the information can be broken down into 64 bits or less chunks up to a maximum number of 64 chunks. In this case, except for the last chunk, all other chunks are registered by SCSI Services as transparent nodes. The last chunk is registered as target path node that would be claimed by either the SCSI Services owned target path dummy driver.

The interface driver cannot use the same `escsi_iftgt_reg_t` structure to register two different target paths because SCSI Services assumes a 1-1 mapping between a `escsi_iftgt_reg_t` structure and a target path. The interface driver must unregister the target from SCSI Services before remapping an existing `escsi_iftgt_reg_t` to a different target.

If the interface driver uses a single I/O object per controller, it must specify `num_iobj` as 1 in `escsi_ifcrtlr_attr_t`. In that case, the `obj_id` field in `escsi_iftgt_attr_t` must be 0 for all targets registered. Interface drivers that use multiple I/O objects can bind each discovered target path to a different I/O object by specifying an `iobj_id` value in `escsi_iftgt_attr_t` during the target registration.

The interface driver must call `escsi_tgt_reg` for all discovered targets on every controller probe, even though the target may have been registered during an earlier invocation of the probe.

RETURN VALUE

None

The target registration status is passed back as `errno` when the target registration callback function is called. This value is one of the following:

- **ESUCCESS** Registration succeeded.
- **EINVAL** Invalid parameter passed or registration failed.

Other `errno` values as returned by `escsi_reg_node` are also applicable.

CONSTRAINTS

This function can be invoked in the ICS context or in a blocking context.

Do not hold any spinlocks when making this call.

Do not call this function outside of the controller probe context. In other words, targets must be registered only as part of a controller probe action.
EXAMPLE

The following example contains a mix of C code and pseudocode.

```c
xxx_ctlr_probe(void *if_ctlr) { 

  Discover target ports/nodes under the controller using
  transport specific probe mechanism

  For every discovered target port {
    /* Allocate and initialize target structures:
       * escsi_iftgt_reg_t
       * escsi_iftgt_attr_t
       * escsi_iftgt_stat_t
    */

    /* Build extended hw paths to target path
     * Specify tp IO node callback routine information :
     * Pointer to callback routine, callback routine arguments,
     * event mask.
    */

    if_tgt_reg->if_tgt_reg_cbfn = xxx_tgt_reg_cbfn;
    if_tgt_reg->addr.c = if_ctlr_reg->addr.c;
    escsi_tgt_reg(iftgt_reg);
  }

  xxx_tgt_reg_cbfn(escsi_iftgt_reg_t *iftgt_reg, int errno)
  {
    If errno is ESUCCESS
      /* The iftgt_reg->addr.t contains the target path
       * instance number
       * iftgt_reg->node contains the pointer to the tp IO node
       * registered with IOI
       * Register target private properties, if any, by calling
       * prop_create()
    */
    else
      /* Free the target registration related structures */
  }

SEE ALSO

prop_create(9F), wsio_reg_node(9F)
```
escsi_tgt_unreg(9F)

NAME
escsi_tgt_unreg - SCSI target unregistration interface

SYNOPSIS
void escsi_tgt_unreg(
    escsi_iftgt_reg_t *iftgt_reg
);

PARAMETERS
iftgt_reg Pointer to a target registration structure allocated by interface driver.

DESCRIPTION
This function is called by interface driver to unregister a target that has been previously registered
with SCSI Services. The interface driver is expected to call this function during an OLD or DLKM
unload operation. After this call, interface driver must deallocate escsi_iftgt_reg_t and
other secondary structures associated with it.
Before calling escsi_tgt_unreg, the interface driver must walk through its internal queues
to check any I/Os currently pending or active on that tgtpath. All such I/Os must be reported
back to SCSI Services as errors.

RETURN VALUE
ESUCCESS Unregistration succeeded.
EINVAL Invalid parameter passed or unregistration failed.

CONSTRAINTS
This call must be made in a blocking context without holding any spinlocks.

EXAMPLE
xxx_tgt_unreg(){
    for every registered target path {
        status = escsi_tgt_unreg(escsi_iftgt_reg_t *iftgt_reg) ;
        if (status == ESUCCESS) {
            deallocate the following structures:
            - escsi_iftgt_reg_t
            - escsi_iftgt_attr_t
            - escsi_iftgt_stat_t
        } else
            VASSERT()
    }
}

SEE ALSO
escsi_ctlr_reg(9F)
**escsi_tm_arg_t(9S)**

**NAME**

escsi_tm_arg_t - Task management structure shared between the SCSI stack and interface drivers.

**SYNOPSIS**

```c
#include <io/escsi_services.h>
```

**DESCRIPTION**

This structure defines the information passed by the SCSI Services to the interface driver through the `escsi_if_sw_t->if_task_mgmt` entry point.

**STRUCTURE MEMBERS**

```c
typedef struct {
    escsi_list_t       list;
    uint64_t           version;
    escsi_task_t       task;
    struct buf *bp;
    int                errno;
    escsi_cbfn_t       cbfn;
    escsi_tm_flags_t   flags;
    intptr_t           rsvd[5];
} escsi_tm_arg_t;
```

- **list**: Specifies the linked list used by interface drivers.
- **version**: Specifies a version of the task management structure.
- **task**: Specifies the task management operation code.
- **bp**: Pointer to the `buf` structure. You can obtain access to the associated I/O object from the `iobj` field in `((escsi_scb_t *)bp->B_SCB)`.
- **errno**: Specifies the `errno` to be returned by the interface driver.
- **cbfn**: Specifies the callback routine and argument to be invoked by the interface driver after the completion of a task management function.
- **flags**: Reserved for SCSI Services. The interface driver must not modify this field.
- **rsvd**: Reserved.

**CONSTRAINTS**

None

**SEE ALSO**

`buf(9S), escsi_ifsw_t(9S), escsi_task_t(9S)`
ESCSI_TRC(9G)

NAME

ESCSI_TRC - Macro to trace up to three 64-bit unsigned values

SYNOPSIS

ESCSI_TRC(
    escsi_trc_buf_t *__tbuf,
    int32_t __nparms,
    intptr_t __v1,
    intptr_t __v2,
    intptr_t __v3
);

PARAMETERS

__tbuf Pointer to a trace buffer.
__nparms Number of actual 64-bit values to trace.
__v1, __v2, __v3 Values to trace.

DESCRIPTION

Interface drivers can use the ESCSI_TRC macro for tracing data values in the controller-specific trace buffer.

You can trace up to three 64-bit unsigned values (either data or pointers). The __nparms parameter indicates how many such values are actually being passed. If it is less than 3, the corresponding __v2 and __v3 values are ignored. Along with these traced values, the macro also logs the logging time, filename, and line number of the caller. This trace information is useful for debugging purposes.

The trace buffer is the controller-level trace buffer allocated earlier by the interface driver and passed to SCSI Services while performing controller registration using escsi_ctlr_reg.

RETURN VALUE

None

CONSTRAINTS

This macro can be called while holding interface driver specific spinlocks.

EXAMPLE

if (status != 0)
    ESCSI_TRC(&ifctlr_reg->trc_info, 2, port_id, status, 0);

SEE ALSO

escsi_trc_init(9F), escsi_trc_uninit(9F), ESCSI_TRC_VERBOSE(9G)
NAME
escsi_trc_buf_t - Trace buffer structure

SYNOPSIS
#include <io/escsi_services.h>

DESCRIPTION
A trace buffer is a circular array of trace record structures.

STRUCTURE MEMBERS
typedef struct {
    escsi_trc_rec_t  *trc_rec;
    uint32_t         trc_head;
    uint32_t         trc_tail;
    uint32_t         tot_trc_recs;
    lock_t           *lock;
} escsi_trc_buf_t;

trc_rec    Pointer to the array of trace records.
rec_head   Specifies the buffer head counter.
trc_tail   Specifies the buffer tail counter.
tot_trc_rec Specifies the total number of trace records.
lock       Pointer to the trace buffer lock.

CONSTRAINTS
None

SEE ALSO
ESCSI_TRC(9G)
NAME

escsi_trc_rec_t  - Trace record data structure

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

The SCSI stack provides a mechanism for tracing function execution. The `escsi_trc_rec_t` data structure defines the basic information of a trace record.

STRUCTURE MEMBERS

typedef struct {
  char                  *fname;
  uint32_t               num_parms      :8,
                          line_num       :24;
  struct timeval         time;
  intptr_t               value1;
  intptr_t               value2;
  intptr_t               value3;
} escsi_trc_rec_t;

fname  Pointer to the source file name.
num_parms Specifies the valid number of trace parameters.
line_num Specifies the line number in the source code file where the tracing is done.
time    Specifies the trace timestamp.
value1  Specifies trace optional parameter 1. Only valid if `num_parms` is greater than or equal to 1.
value2  Specifies trace optional parameter 2. Only valid if `num_parms` is greater than or equal to 2.
value3  Specifies trace optional parameter 3. Only valid if `num_parms` is greater than or equal to 3.

CONSTRAINTS

None

SEE ALSO

ESCSI_TRC(9G), escsi_trc.but_t(9S), ESCSI_TRC_VERBOSE(9G)
ESCSI_TRC_VERBOSE(9G)

NAME

ESCSI_TRC_VERBOSE - Macro to trace up to three 64-bit unsigned values, if verbose option is on

SYNOPSIS

ESCSI_TRC_VERBOSE(
    escsi_trc_buf_t *__tbuf,
    int32_t __nparms,
    intptr_t __v1,
    intptr_t __v2,
    intptr_t __v3
);

PARAMETERS

__tbuf Pointer to a trace buffer.
__nparms Number of actual 64-bit values to be traced.
__v1, __v2, __v3 Values to trace.

DESCRIPTION

Interface drivers can use the ESCSI_TRC_VERBOSE macro for tracing data values in the controller-specific trace buffer.

This macro is exactly identical to ESCSI_TRC, but performs the tracing only when the caller source code as well as SCSI Services are called with __ESCSI_DEBUG compilation switch enabled. This macro can be used to collect additional debugging information in a debug kernel that are otherwise not required in a normal performance kernel. The decision on whether to use ESCSI_TRC or ESCSI_TRC_VERBOSE is up to the caller (interface driver), depending on the debugging needs.

RETURN VALUE

None

CONSTRAINTS

This macro can be called while holding interface driver specific spinlocks.

EXAMPLE

ESCSI_TRC_VERBOSE(&ifctrlr_reg->trc_info, 2, port_id, status, 0);

SEE ALSO

ESCSI_TRC(9G), escsi_trc_init(9F), escsi_trc_uninit(9F)
escsi_type_t(9S)

NAME
escsi_type_t - SCSI object types

SYNOPSIS
#include <io/escsi_services.h>

DESCRIPTION
This structure describes the types of a SCSI objects. It is used in requests where it is necessary to
qualify the type of SCSI object identified by an opaque handle (for example, I/O node handle).

STRUCTURE MEMBERS

typedef enum {
    ESCSI_LUN         = 0x01,    /* object is a LUN */
    ESCSI_LPT         = 0x02,    /* object is a lun path */
    ESCSI_CTLR        = 0x03,    /* object is a controller */
    ESCSI_TP          = 0x04,    /* object is a target path */
    ESCSI_GLOBAL      = 0x05     /* initiator node */
} escsi_type_t;

CONSTRAINTS
None

SEE ALSO
escsi_node_to_addr(9F)
NAME
escsi_vidpid_reg - Register VID and PID information with the escsi_ddsw_t structure.

SYNOPSIS
void escsi_vidpid_reg(
    escsi_ddsw_t *ddsw,
    escsi_devid_t *vidpid
);

PARAMETERS
ddsw Pointer to the device driver switch table.
vidpid Pointer to the VID and PID information to be associated with the device driver switch table.

DESCRIPTION
The escsi_vidpid_reg routine registers VID and PID information with the escsi_ddsw_t structure. Based on the VID and PID registered, SCSI Services binds the class driver to a specific device matching the class driver VID and PID information provided. If the driver has several pdt/vid/pid triplets to register, the driver_install routine must call escsi_vidpid_reg multiple times, passing in a escsi_devid_t structure each time.

Call this routine after the call to escsi_dd_reg because escsi_dd_reg zeroes out the triplet information in the escsi_ddsw_t structure.

This service is optional.

RETURN VALUES
None

CONSTRAINTS
Must be invoked in the driver install entry point after the call to escsi_dd_reg.

EXAMPLE
None

SEE ALSO
escsi_dd_reg(9F), escsi_ddsw_t(9S)
escsi_vsema(9F)

NAME
escsi_vsema - Release an eSCSI semaphore.

SYNOPSIS
void escsi_vsema(
    lock_t *lock,
    escsi_bsema_t *sema
);

PARAMETERS
lock  Pointer to the lock protecting the escsi_bsema_t semaphore.
sema  Pointer to the escsi_bsema_t semaphore.

DESCRIPTION
The escsi_vsema routine releases the eSCSI semaphore.

RETURN VALUES
None

CONSTRAINTS
The lock parameter is mandatory.

EXAMPLE
None

SEE ALSO
escsi_init_sema(9F), escsi_owns_sema(9F), escsi_psema(9F)
esctl_io_t(9S)

NAME

esctl_io_t - Structure that describes the I/O request being issued via the SIOC_IO_EXT ioctl.

SYNOPSIS

#include <io/escsi_services.h>

DESCRIPTION

The SIOC_IO_EXT ioctl can be issued by user or kernel application.

NOTE: Explicit support of SIOC_IO_EXT for 32-bit mode is not necessary as the esctl_io_t definition is 32-bit/64-bit neutral.

STRUCTURE MEMBERS

typedef struct {
    int version;
    escsi_sctl_io_flags_t flags;
    int max_msecs;
   .uint32_t cdb_length;
    uint32_t data_length;
    ptr64_t data;
    union sense_data sense;
    escsi_hw_path_t lpt_hwp;
    uint32_t data_xfer;
    uint32_t sense_xfer;
    uint32_t cdb_status;
    uint32_t sense_status;
    uint8_t cdb[ESCSI_MAX_CDB_LEN];
    uint32_t rsvd[32];
} esctl_io_t;

version

    Specifies the structure version.

flags

    Specifies esctl_io flags.

max_msecs

    Specifies the timeout for the I/O (in milliseconds). The caller can also indicate that the default timeout value be used or that the I/O must not be timed.

cdb_length

    Specifies the size of the cdb.

data_length

    Specifies the size of the data buffer.

data

    Specifies the data buffer pointer. On return, for read I/O operations this buffer contains the data read from the device.

sense

    Specifies the sense data buffer. On return, this contains sense information from the device when CHECK CONDITION is returned.

lpt_hwp

    Specifies the hardware path of the lunpath to be used to issue the I/O.

data_xfer

    On return, specifies the number of bytes transferred from the device for read I/O operations.

sense_xfer

    On return, specifies the number of bytes of sense data copied into the sense buffer.

cdb_status

    On return, specifies I/O status.

sense_status

    On return, specifies sense status. This field is valid if cdb_status indicates CHECK CONDITION.

cdb

    Specifies the CDB. This must be the last field before the rsvd field in this structure. This is intentional in order to allow for the handling of variable or increasing sized cdb's in the future.
rsvd
Reserved for future use.

CONSTRAINTS
None

SEE ALSO
escsi_hw_path_t(9S), escsi_sctl_io_flags_t(9S)
ipf_efi2hw_data(9S)

NAME

ipf_efi2hw_data - Structure to support the IOC_EFI2HW ioctl.

SYNOPSIS

#include <machine/plat/misc/kern_drv_iface_pdk.h>

DESCRIPTION

This structure contains data definitions for the IOC_EFI2HW ioctl.

STRUCTURE MEMBERS

struct ipf_efi2hw_data {
    EFI_DEVICE_PATH *node;
    hw_path_t        *hw;
    uint64_t          flags;
};

node Pointer to the EFI node to convert to a hardware path element.

hw Pointer to the hardware path structure being constructed.

flags Reserved.

SEE ALSO

driver_ioctl(9F), hw_path_t(9S)
NAME

ipf_efi2str_data - Structure to support the IOC_EFI2STR ioctl.

SYNOPSIS

#include <machine/plat/misc/kern_drv_iface_pdk.h>

DESCRIPTION

This structure contains data definitions for the IOC_EFI2STR ioctl.

STRUCTURE MEMBERS

struct ipf_efi2str_data {
    EFI_DEV_PATH   node;
    char           buffer[EFI_MAXIMUM_VARIABLE_SIZE];
    uint64_t       str_length;
    uint64_t       flags;
};

node         EFI node being converted to a hardware path element.
buffer       Buffer being filled with EFI string data.
str_length   Size (in bytes) of the string the driver placed in buffer.
flags        Reserved

SEE ALSO

driver_ioctl(9F)
ipf_hw2efi_data(9S)

NAME

eipf_hw2efi_data -- Structure to support the IOC_HW2EFI ioctl.

SYNOPSIS

#include <machine/plat/misc/kern_drv_iface_pdk.h>

DESCRIPTION

This structure contains data definitions for the IOC_HW2EFI ioctl.

STRUCTURE MEMBERS

struct struct ipf_hw2efi_data {
    hw_path_t *hw;
    uint8_t   *buffer;
    size_t     buf_length;
    size_t     node_length;
    uint64_t   flags;
};

hw

Pointer to the entire hardware path structure being converted to EFI path. At
the point the ioctl is called, hw->last_index is pointed to the HBA node.

buffer

Pointer to current position in the buffer being filled with EFI node data. The
interface driver places its portion of the device path after the HBA node up
to and including the LUN.

buf_length

Size (in bytes) of the unused portion of buffer, which is used to store the
generated EFI path.

node_length

Size (in bytes) of the EFI node the driver conversion routine just created.

flags

Reserved.

SEE ALSO

driver_ioctl(9F), hw_path_t(9S)
NAME
SIOC_ATTR - ioctl to set and get SCSI attributes.

SYNOPSIS
#include <sys/scsi.h>

DESCRIPTION
The SIOC_ATTR ioctl sets and gets values of SCSI stack global attributes or attributes of an instance of a SCSI object. Class drivers can use this ioctl to set and get class driver-specific object attributes. Implementing this ioctl enables a user to perform management operations on class driver SCSI object attributes using the scsimgr command line interface.

The sioc_attr_t encapsulates the following types:

escsi_id_info_t     A SCSI object identifier. This encapsulates the following types:
                  escsi_id_type_t   The type of a SCSI object identifier.
                  escsi_id_scope_t The SCSI object identifier scope. SCSI Services and class drivers can maintain information about a SCSI object. To enable class drivers to issue management ioctls on /dev/escsi, the ESCSI_IDS_DRV scope specifies to forward the ioctl issued on /dev/escsi to the class driver d_ioctl entry point.
                  escsi_id_t The SCSI object identifier value: dev_t, hardware path, or name (DDR entry name or driver name). The driver name references the driver globally and enables a class driver to have attributes global to the driver. The DDR entry name references a certain scope of class driver to have device specific attributes.

escsi_attr_opt_t The type of operation performed on the attribute: get, set, or save.

escsi_attr_t A SCSI attribute. It encapsulates:
                  escsi_attr_flag_t The degree of persistency of a SCSI attribute: default, saved, or current.
                  escsi_attr_type_t The attribute value type.
                  escsi_attr_val_t A SCSI attribute value. It encapsulates escsi_user_dat_t, representing user data.

The escsi_id_info_t structure is defined as follows:

typedef enum {
    ESCSI_ID_NONE    = 0,
    ESCSI_ID_DEV     = 1,
    ESCSI_ID_CL_INST = 2,
    ESCSI_ID_NAME    = 3,
    ESCSI_ID_HWP     = 4
} escsi_id_type_t;

ESCSI_ID_NONE No identifier type. Use this to get global objects.
ESCSI_ID_DEV Object identified by a dev_t.
ESCSI_ID_CL_INST  Reserved.
ESCSI_ID_NAME   Object is identified by a null-terminated string.
ESCSI_ID_HWP    Object is identified by hardware path.

The escsi_id_t structure is defined as follows:

```c
#define ESCSI_ID_NAME_SZ 256 /* Maximum size of a name identifier */

typedef union {
    dev_t dev;
    escsi_class_t ci;
    escsi_hw_path_t hwp;
    char name[ESCSI_ID_NAME_SZ];
} escsi_id_t;

dev    Specifies the dev_t identifier.
ci     Reserved.
hwp    Specifies the hardware path identifier.
name   Specifies the name identifier. For example, DDR entry name.
```

The escsi_id_scope_t structure is defined as follows:

```c
typedef enum {
    ESCSI_IDS_GEN = 1,
    ESCSI_IDS_DRV = 2
} escsi_id_scope_t;

ESCSI_IDS_GEN     The identifier scope is SCSI Services.
ESCSI_IDS_DRV     The identifier scope is the class driver. Therefore, the ioctl is directed to the
                  class driver.
```

The escsi_id_info_t structure is defined as follows:

```c
typedef struct escsi_id_info {
    escsi_id_type_t    type;
    escsi_id_scope_t   scope;
    escsi_id_t         id;
    uint64_t           rsvd[4];
} escsi_id_info_t;

type     Specifies the identifier type.
scope    Specifies the identifier scope.
id       Specifies the known identifier.
rsvd     Reserved.
```

The escsi_attr_op_t structure is defined as follows:

```c
typedef enum {
    ESCSI_GET_VAL   = 1,
    ESCSI_SET_VAL   = 2,
    ESCSI_SAVE_VAL  = 3
} escsi_attr_op_t;

ESCSI_GET_VAL     Get the value(s) of the attribute(s).
ESCSI_SET_VAL     Set the current value(s) of the attribute(s).
ESCSI_SAVE_VAL    Save the default value(s) of the attribute(s).
```

The escsi_attr_flag_t structure is defined as follows:

```c
typedef enum {
    ESCSI_CURRENT    = 0x0001,
    ESCSI_SAVED      = 0x0002,
    ESCSI_DEFAULT    = 0x0004
} escsi_attr_flag_t;

ESCSI_CURRENT     Current attribute value
ESCSI_SAVED     Saved attribute value
ESCSI_DEFAULT   Default attribute value

The escsi_attr_type_t structure is defined as follows:

typedef enum {
    ESCSI_BOOL        = 1,    /* Boolean */
    ESCSI_INT8        = 2,    /* 8 bit integer */
    ESCSI_UINT8       = 3,    /* 8 bit unsigned integer*/
    ESCSI_INT16       = 4,    /* 16 bit signed integer */
    ESCSI_UINT16      = 5,    /* 16 bit unsigned integer */
    ESCSI_INT32       = 6,    /* 32 bit integer */
    ESCSI_UINT32      = 7,    /* 32 bit unsigned integer */
    ESCSI_INT64       = 8,    /* 64 bits integer */
    ESCSI_UINT64      = 9,    /* 64 bit unsigned integer */
    ESCSI_STRING      = 10,   /* NULL terminated string */
    * The maximum size of the string is still limited though (see
    * escsi_attribute_t */
    ESCSI_USER        = 11    /* user defined data of given size. The maximum size
    * supported is limited (see * escsi_attribute_val_t) */
} escsi_attr_type_t;

The escsi_user_dat_t structure is defined as follows:

typedef struct {
    uint    size;
    uint8_t dat[ESCSI_A_USER];
} escsi_user_dat_t;

size    Specifies the size (in bytes) of the user data.
dat     User data buffer.

The escsi_attribute_val_t structure is defined as follows:

typedef union escsi_attribute_val {
    int8_t             i8;      /* 8 bit integer */
    uint8_t            u8;      /* 8 bit unsigned int */
    int16_t            i16;     /* 16 bit integer */
    uint16_t           u16;     /* 16 bit unsigned int */
    int32_t            i32;     /* 32 bit integer */
    uint32_t           u32;     /* 32 bit unsigned integer */
    escsi_bool_t       bl;      /* Boolean */
    int64_t            i64;     /* 64 bit integer */
    uint64_t           u64;     /* 64 bit unsigned integer */
    char               str[ESCSI_A_STRING];  /* null terminated string */
    escsi_user_dat_t   user;    /* user defined data with a
    * given size */
} escsi_attribute_val_t;

The escsi_attribute_t structure is defined as follows:

typedef struct {
    escsi_attribute_flag_t flags;
    escsi_attribute_type_t type;
    char                 name[ESCSI_A_NAME];
    escsi_attribute_val_t cur_val;
    escsi_attribute_val_t sav_val;
    escsi_attribute_val_t def_val;
    int status;
} escsi_attribute_t;

flags      Flag representing current, saved, or default value of the attribute.
type Specifies the attribute type.
name Specifies the attribute name.
cur_val Specifies the current value of the attribute.
sav_val Specifies the saved value of the attribute.
def_val Specifies the default value of the attribute.
status Specifies one of the following results of operation on attribute:
0     Success.
EINVAL Invalid value.
ENOENT Unknown attribute.
ENOTSUP Operation not supported. For example, set on read only attribute.

The sioc_attr_t structure is defined as follows:
typedef struct {
    escsi_id_info_t    id;
    escsi_attr_op_t    op;
    uint32_t           num_attr;
    escsi_attr_t       attrs[ESCSI_MAX_ATTR];
    uint64_t           rsvd[4];
} sioc_attr_t;

id Specifies the SCSI object identifier.
op Specifies the operation to perform.
num_attr Specifies the number of valid entries in the attrs array.
attrs Specifies an array of attributes on which to perform the operation.
rsvd Reserved.

RETURN VALUES
0     Success.
EINVAL Invalid value.
EACCESS Insufficient privilege for the attempted operation.

CONSTRAINTS
This ioctl is only supported on the /dev/escsi management device file. When this ioctl is used for driver-specific attributes, it is forwarded to the d_ioctl entry point.

EXAMPLE
int mydriver_ioctl(dev_t ldev, int cmd, caddr_t udata_ptr, int flags) {
    /*
     * If dev_t is management dev_t, call common
     * management routine
     */
    if (ESCSI_IS_MGMT_DEV(ldev) == ESCSI_TRUE) {
        return mydriver_mgmt_ioctl(ldev, cmd, udata_ptr, flags);
    }
    .
    .
}

int mydriver_mgmt_ioctl(dev_t dev, int cmd, caddr_t udata_ptr, int flags) {
    int error = ESUCCESS;
    escsi_id_info_t *id;
escsi_info_t *escsi_info = NULL;

id = (escsi_id_info_t *) udata_ptr;

switch(cmd) {
    case SIOC_ATTR :
        attr = (sioc_attr_t *) udata_ptr;

        if(id->type == ESCSI_ID_NAME) {
            if < name is a DDR entry name > {
                if(attr->op == ESCSI_GET_VAL) {
                    error = mydrv_get_ddr_attr(attr->id.id.name,
                                            attr->attrs, attr->num_attr);
                } else {
                    /* Set DDR attr */
                    .
                    .
                    .
                }
            } else {
                /* driver name */
                escsi_info = escsi_global.escsi_info;
            }
        } else {
            /* It is a SCSI object id: lookup it up */
            escsi_info = escsi_lookup_obj(id);
            if(escsi_info == (escsi_info_t *)NULL) {
                return EINVAL;
            }

            switch(escsi_info->type) {
                case ESCSI_GLOBAL:
                    /* Get or set class driver private global attributes */
                    .
                    .
                    .
                    break;

                case ESCSI_LUN:
                    /* Get or set class driver private LUN attribute */
                    if(attr->op == ESCSI_GET_VAL) {
                        .
                        .
                        .
                    } else {
                        error = mydrv_set_lun_attr((escsi_lun_t *) escsi_info,
                                            attr->attrs, attr->num_attr, attr->op);
                        break;
                    }

                case ESCSI_LPT:
                    /* Get or set class driver private lunpath attribute */
                    my_lpt=(my_lpt_t*)(((escsi_lpt_t*)escsi_info)->dd_lpt);
                    .
                    .
                    .
                    /*
                     * The lunpath object is deletable. To protect
                     * against its deletion, escsi_lookup_obj() invoked
                     * ESCSI_HOLD() when the object was looked up.
                     * Since we are done with the operation on the
                     * escsi_lpt_t object, release it.
                     */
                    ESCSI_RELE(lpt, ESCSI_FALSE)
            }
        }
}
mydrv_get_ddr_attr(char *name, escsi_attr_t *attrs, uint32_t num_attr)
{
    escsi_krid_t krid;
    escsi_kr_key_t key;
    uint32_t buf1;
    uint32_t i;
    int error;
    kr_type_t kr_type;
    escsi_attr_t *attr = attrs;

    /* open krs node corresponding to global */
    krid.id_type = ESCSI_NAME;
    strcpy(krid.kr_info.name, name);

    error = escsi_kr_open(&krid, ESCSI_GLOBAL, ESCSI_KR_NOFLAG, &key);

    /* DDR entry may not exist. So process error */
    if(error != ESUCCESS) {
        return ENOENT;
    }

    for(i=0; i< num_attr; i++, attr++)
    {
        buf1 = sizeof(attr->cur_val.u32);
        attr->type = ESCSI_UINT32;
        escsi_kr_get_settable_attr(&key, attr->name,
            (void *) &attr->cur_val.u32,
            (void *) &attr->def_val.u32,
            (void *) &attr->sav_val.u32, &buf1,
            &attr->flags, &attr->status, &kr_type);

        continue; /* Unknown attribute. Return ENOENT */
        attr->status = ENOENT; */
    } /* (i=0; i< num_attr; i++, attr++) */

    escsi_kr_close(&key);
    return ESUCCESS;
}

/* Assuming an attribute MYDRV_LUN_ATTR of type uint32_t */
#define MYDRV_LUN_ATTR   "mydriver lun_attribute"

/* default, min and max value of the attribute */
#define DEF_MYDRV_LUN_ATTR_30
#define MIN_MYDRV_LUN_ATTR_SECS 0
#define MAX_MYDRV_LUN_ATTR xFFFFFFFF
int
mydrv_set_lun_attr() {
    escsi_krid_t krid;
    escsi_kr_key_t key;
    uint32_t buf2;
    uint64_t buf1;
    uint32_t i;


int error;
kr_type_t kr_type;
escsi_kr_flags_t kr_flags = 0;
mydrv_lun_t *mydrv_lun = (mydrv_lun_t *) lun->dd_lun;
uint32_t *cached_val;
uint32_t cur_lbal;

for (i=0; i < num_attr; i++, attr++) {
    if ((op == ESCSI_SAVE_VAL) && (attr->flags & ESCSI_DEFAULT)) {
        kr_flags |= ESCSI_KR_DEFAULT;
    } else {
        kr_flags &= ~ESCSI_KR_DEFAULT;
    }

    if (strcmp(attr->name, MYDRV_ATTR_) == 0) {
        /* Validate the range if it is saveable */
        if (((kr_flags & ESCSI_KR_DEFAULT) == 0) &&
          (attr->cur_val.u32 > MAX_MYDRV_LUN_ATTR)) {
            attr->status = EINVAL;
            continue;
        }
    }

    buf1 = (uint64_t)sizeof(attr->cur_val.u32);
    error = escsi_kr_set_attr(&key, attr->name, kr_flags, buf1,
                              (void *)&attr->cur_val.u32, KR_VTYPE_UINT32);
    attr->status = error;
    continue;
}

    /* Set the other class driver LUN attributes */
    
}
escsi_kr_close(&key);
return ESUCCESS;

SEE ALSO

ESCSI_IS_MGMT_DEV(9G), ESCSI_RELE(9G), escsi_kr_close(9F), escsi_kr_get_settable(9F),
escsi_kr_lookup_obj(9F), escsi_kr_open(9F), escsi_kr_set_attr(9F)
**SIOC_CLEAR_STAT(9G)**

**NAME**

SIOC_CLEAR_STAT - ioctl to clear the statistics of a class driver SCSI object.

**SYNOPSIS**

```c
#include <sys/escsi_stack.h>
```

**DESCRIPTION**

The **SIOC_CLEAR_STAT** ioctl clears the statistics of a SCSI object (global, LUN, or lunpath) owned by the class driver. Class drivers can keep some error condition statistics for each one of their SCSI objects such as a global object, LUN object, or lunpath object. Under certain circumstances, it might be necessary to clear an object statistics. Implementing this ioctl enables a user to clear class driver statistics using the `scsimgr` command line interface.

This ioctl takes `escsi_id_info_t` as a parameter. See **SIOC_ATTR(9G)** for a definition of this type.

**RETURN VALUES**

- **0**  
  Success.
- **EINVAL**  
  Invalid value.
- **EACCESS**  
  Insufficient privilege for the attempted operation.

**CONSTRAINTS**

To protect against deletion of objects (**escsi_lpt_t**), the **escsi_lookup_obj** object lookup service invokes **ESCSI_HOLD** on the object. This function must invoke **ESCSI_RELE** on the object before returning.

**EXAMPLE**

```c
int mydriver_ioctl(dev_t ldev, int cmd, caddr_t udata_ptr, int flags)
{
    /*
     * If dev_t is management dev_t, call common
     * management routine
     */
    if(ESCSI_IS_MGMT_DEV(ldev) == ESCSI_TRUE) {
        return mydriver_mgmt_ioctl(ldev, cmd, udata_ptr, flags);
    }
    .
    .
}
```

```c
int mydriver_mgmt_ioctl(dev_t dev, int cmd, caddr_t udata_ptr, int flags)
{
    int       error = ESUCCESS;
    escsi_id_info_t *id;
    escsi_info_t *escsi_info = NULL;
    id = (escsi_id_info_t *) udata_ptr;

    switch(cmd) {
    case SIOC_CLEAR_STAT:
        error = mydrv_clear_stat(id);
        break;
    .
    .
    }
}
```
return errno;
}

int
mydrv_clear_stat(escsi_id_info_t *id)
{
    escsi_info_t *escsi_info = NULL;
    mydrv_lpt_t *mydrv_lpt;
    int errno = ESUCCESS;

    if(id->type != ESCSI_ID_NAME) {
        /*
         * get escsi_info_t from the object identifier and
         * hold the object if it can be deleted (lpt, tp, ctlr)
         */
        escsi_info = escsi_lookup_obj(id);
    }

    if(escsi_info == (escsi_info_t *) NULL) {
        return EINVAL;
    }

    switch(escsi_info->type) {
    case ESCSI_LUN :
        /* Clear class driver LUN stats */
        .
        .
        .
        break;

    case ESCSI_LPT:
        mydrv_lpt = (mydrv_lpt_t *)((escsi_lpt_t *)escsi_info)->dd_lpt);
        bzero(&mydrv_lpt->stat, sizeof(mydrv_lpt->stat));
        ESCSI_RELE(escsi_info, ESCSI_FALSE);
        break;
        .
        .
        .

    return errno;
    }

**SEE ALSO**

*ESCSI_HOLD(9G), ESCSI_RELE(9G), escsi_lookup_obj(9F), SIOC_ATTR(9G)*

1090 SCSI Reference Pages
NAME

SIOC_DDR - ioctl to add or delete a DDR entry.

SYNOPSIS

#include <sys/escsi_stack.h>

DESCRIPTION

The SIOC_DDR ioctl enables a user to create or delete a DDR entry from the scsimgr command line utility. The name of the DDR entry is specified in the sioc_ddr_t->id ioctl parameter.

This ioctl takes the following parameters:

typedef enum sioc_ddr_op {
    SIOC_DDR_ADD     = 0x01,
    SIOC_DDR_DEL     = 0x02,
} sioc_ddr_op_t;

sioc_ddr_op_t

SIOC_DDR_ADD Add a DDR entry.

SIOC_DDR_DEL Delete a DDR entry.

typedef  struct {
    escsi_id_info_t id;
    sioc_ddr_op_t   op;
    uint64_t        rsvd[4];
} sioc_ddr_t;

id Name of the DDR entry. For information on escsi_id_info_t, see SIOC_ATTR(9G).

op DDR entry operation: add or delete.

rsvd Reserved.

RETURN VALUES

0 Success.
EINVAL Invalid value.
EACCESS Insufficient privilege for the attempted operation.

CONSTRAINTS

The escsi_id_info_t structure is assumed to be the common header of all management ioctls. Assume the object identifier has a driver scope (ESCSI_IDS_DRV).

EXAMPLE

int mydriver_ioctl(dev_t ldev, int cmd, caddr_t udata_ptr, int flags) {

    /*
     * If dev_t is management dev_t, call common
     * management routine
     */
    if(ESCSI_IS_MGMT_DEV(ldev) == ESCSI_TRUE) {
        return mydriver_mgmt_ioctl(ldev, cmd, udata_ptr, flags);
    }
    .
    .
}

int
mydriver_mgmt_ioctl(dev_t dev, int cmd, caddr_t udata_ptr, int flags)
```c
{
    int error = ESUCCESS;
    escsi_id_info_t *id;
    escsi_info_t *escsi_info = NULL;

    id = (escsi_id_info_t *) udata_ptr;

    switch(cmd) {
    case SIOC_DDR:
        ddr = (sioc_ddr_t *) udata_ptr;
        krid.id_type = ESCSI_NAME;
        strcpy(krid.kr_info.name, id->id.name);
        if(ddr->op == SIOC_DDR_ADD) {

            /* Check that entry does not already exists */
            error = escsi_kr_open(&krid, ESCSI_GLOBAL,
                                  ESCSI_KR_OPEN_OBJ, &key);
            if (error == ESUCCESS) {
                /* Entry already exists. return EEXIST */
                escsi_kr_close(&key);
                error = EEXIST;
                break;
            }

            /* Try to create the new DDR entry persistently */
            error = escsi_kr_open(&krid, ESCSI_GLOBAL,
                                  ESCSI_KR_CREATE| ESCSI_KR_PERSISTENT,&key);
            /* Entry added successfully. Close it */
            escsi_kr_close(&key);
        } else {
            /* DDR entry deletion */
            /* Perform some driver specific DDR entry checking */
            ...

            /* If the entry does not exist, return ENOENT. */
            if((error=escsi_kr_open(&krid, 0, ESCSI_KR_OPEN_OBJ, &key)) !=
               ESUCCESS) {
                error = ENOENT;
                break;
            }

            /* Delete the entry */
            error = escsi_kr_delete(&key);
        }
    }
}

SEE ALSO
ESCSI_IS_MGMT_DEV(9G), escsi_kr_close(9F), escsi_kr_delete(9F), escsi_kr_open(9F)
```
SIOC_DISABLE(9G)

NAME

SIOC_DISABLE - ioctl to disable access to a SCSI object.

SYNOPSIS

#include <sys/escsi_stack.h>

DESCRIPTION

The SIOC_DISABLE ioctl enables a user to disable a SCSI object managed by a specific class driver using the scsimgr command line utility. The class driver must block I/O requests trying to access to this object. This object could be a lunpath or a LUN.

This ioctl takes escsi_id_info_t as a parameter. For a definition of this type, see SIOC_ATTR(9G).

The type field can be one of the following:
ESC_SI_ID_DEV The known identifier is a dev_t. This identifier is only valid for LUN.
ESC_SI_ID_EHWP The known identifier is an hardware path. It is valid for both LUN and lunpath.

The scope field must be ESCSI_IDS_DRV.

The id field must be populated with the correct identifier.

RETURN VALUES

0 Success.
EINVAL Invalid value.
EACCESS Insufficient privilege for the attempted operation.

CONSTRAINTS

If the object is a LUN, the LUN semaphore must be held across the disable operation.

If the object is a lunpath, ESCSI_HOLD is internally performed upon looking up the object with escsi_lookup_obj. You must invoke ESCSI_RELE explicitly upon completion of the disable operation.

EXAMPLE

None

SEE ALSO

SIOC_ATTR(9G), SIOC_ENABLE(9G)
NAME
SIOC_ENABLE - ioctl to enable access to a SCSI object.

SYNOPSIS
#include <sys/escsi_stack.h>

DESCRIPTION
The SIOC_ENABLE ioctl reenables access to a SCSI object that was suspended either internally by a class driver or externally by the SIOC_DISABLE ioctl.
The class driver must reallow I/O requests sent the suspended object. This object could be a lunpath or a LUN.
This ioctl takes sioc_enable_t as a parameter. This structure is defined as follows:
typedef struct {
    escsi_id_info_t id;
    uint32_t susp_rsn;
    uint64_t rsvd[2];
} sioc_enable_t;

id Specifies the SCSI object identifier. For a description of escsi_id_info_t, see SIOC_ATTR(9G). The id.type field can be one of the following:
ESCSI_ID_DEV The known identifier is a dev_t. This identifier is only valid for LUN.
ESCSI_ID_EHWP The known identifier is a hardware path. This is valid for both LUN and lunpath.
The id.scope field must be ESCSI_IDS_DRV.
The id.id field must be populated with the correct identifier.
susp_rsn Specifies the suspension reason.
rsvd Reserved.

RETURN VALUES
0 Success.
EINVAL Invalid value.
EACCESS Insufficient privilege for the attempted operation.

CONSTRAINTS
Class drivers must only support enabling LUN or lunpath objects.
If the object is a LUN, the LUN semaphore must be held across the resume operation.
If the object is a lunpath, ESCSI_HOLD is internally performed upon looking up the object with escsi_lookup_obj. You must invoke ESCSI_RELE explicitly upon completion of the resume operation.

EXAMPLE
None

SEE ALSO
escsi_lookup_obj(9F), SIOC_DISABLE(9G)
xxx_if_ctrl_probe(9E)

NAME
xxx_if_ctrl_probe - Controller probe entry point

SYNOPSIS
void xxx_if_ctrl_probe(
    void *if_ctlr
);

PARAMETERS
if_ctlr Pointer to an interface-specific controller structure.

DESCRIPTION
SCSI Services calls the xxx_if_ctrl_probe entry point to probe a controller under the interface
driver. As part of the probe, interface driver is expected to discover targets underneath the
controller and register these target paths with SCSI Services. In the function name, xxx represent
the actual interface driver function entry point, which must be specified in the escsi_ifsw_t
structure provided by the interface driver.

This entry point is asynchronous and SCSI Services provides a callback function
escsi_if_ctlr_prb_cbfn, which must be called back by interface driver on completion of
the controller probe. The parameters to this callback function are if_ctlr and the errno value
indicating the probe completion status.

RETURN VALUE
None

EXAMPLE
None

CONSTRAINTS
None

SEE ALSO
escsi_if_ctlr_prb_cbfn(9F)
xxx_if_leg_hwp_build(9E)

NAME

xxx_if_leg_hwp_build - Legacy hardware path build entry point

SYNOPSIS

```c
void xxx_if_leg_hwp_build(
    escsi_ifctlr_reg_t *ifctlr_reg,
    escsi_iftgt_reg_t *iftgt_reg,
    uint32_t num_lunids,
    escsi_lunid_t *lunid_list
);
```

PARAMETERS

- `ifctlr_reg` Pointer to a `escsi_ifctlr_reg_t` structure.
- `iftgt_reg` Pointer to a `escsi_iftgt_reg_t` structure.
- `num_lunids` Number of LUN ids in the list.
- `lunid_list` List of successfully probed LUN ids, including target LUN 0.

DESCRIPTION

SCSI Services calls the `xxx_if_leg_hwp_build` entry point to build legacy format hardware paths for each of the LUNs specified in the list of LUN ids provided. Interface drivers are expected to use this LUN id list to build legacy hardware paths for each of the LUNs and return back to SCSI Services. In the function name, `xxx` represent the actual interface driver function entry point, which must be specified in the `escsi_ifsw_t` structure provided by the interface driver.

In the context of `if_leg_hwp_build`, the interface driver does the following:

- In the case where several LUN addressing methods are used in the LUN id list, select one LUN addressing method as in versions of HP-UX prior to 11i v3. For instance, FibreChannel uses the LUN addressing method selection scheme of FCP CDIO in HP-UX 11i v2. FCP CDIO selects the LUN ids of the same addressing method type by choosing the following addressing method selection order:
  1. Flat space (or Volume set) (Address Method: b01)
  2. Logical Unit (Address Method: b10)
  3. Peripheral Device (Address Method: b00)

After the addressing method is chosen, the interface driver builds legacy hardware paths to the LUN ids whose LUN addressing method matches the selected LUN addressing method.

- Build the legacy hardware paths to legacy target, legacy (virtual) busses, legacy (virtual) target paths, and legacy (virtual) lunpaths.
- Register the built legacy hardware paths with I/O infrastructure and the SCSI Services using the following services:
  - `wsio_reg_legacy` To directly register a legacy target path hardware path with I/O infrastructure, and a virtual legacy bus hardware path with I/O infrastructure. For example, FC <HBA hwpath>:<Domain>:<Area>:<Port>: 0/4/1/0.8.0.15.
  - `escsi_reg_leg_bus` To register a legacy virtual bus with the SCSI Services. For example, FC virtual bus fcparray at 0/4/1/0.8.0.15.0.
  - `escsi_reg_leg_tgt` To register a legacy (virtual) target path with I/O infrastructure and the SCSI Services. For example, FC virtual target path at 0/4/1/0.8.0.15.0.0.
To register a legacy (virtual) lunpath with I/O infrastructure and the SCSI Services. For example, FC virtual lunpath at 0/4/1/0.8.0.15.0.0.1.

To get some Inquiry Data and Evpd page 0x80 data associated with a LUN id.

**RETURN VALUE**

None

**CONSTRAINTS**

The interface driver cannot sleep in the context of the if_leg_hwp_build entry point.

**EXAMPLE**

```c
if_bld_leg_hw_path(
    escsi_ifctrl_reg_t      *ifctrl_reg,
    escsi_iftgt_reg_t       *iftgt_reg,
    uint32_t                      num_lunids,
    escsi_lunid_t            *lunid_list)
{
    wsio_legacy_info_t          target_leg_info;
    wsio_legacy_info_t          bus_leg_info;
    escsi_lunid_t              *lunid_elt;
    escsi_addr_t                addr;
    escsi_addr_t                new_lpt_addr;
    escsi_get_leg_lun_info_t    lpt_info;
    escsi_bool_t                leg_bus_map;
    uint8_t                     leg_bus_id;
    uint8_t                     leg_tgt_id;
    uint8_t                     leg_lun_id;
    uint8_t                     leg_bus_inst;
    escsi_reg_leg_bus_t         leg_bus_reg;
    escsi_reg_leg_tgt_t         leg_tgt_reg;
    escsi_reg_leg_lun_t         leg_lun_reg;
    hw_path_t                   leg_bus_hw_path;
    hw_path_t                   leg_tgt_hw_path;
    hw_path_t                   leg_lun_hw_path;
    wsio_ret_code_t             ret;
    int                         status;

    /* Build legacy target wsio_legacy_info_t and register it with
     * IOI by invoking wsio_reg_legacy()
     */
    target_leg_info.hw_path :=
        FC : <ctlr hwpth>.<domain>.<area>.<port>
        iSCSI: /255/<virtual krio bus/<session instance>
    target_leg_info.drv_info := none if it is a transparent hw path
    target_leg_info.type := T_CDIO_PRIVATE
    target_leg_info.id := none if it is a transparent hw path
    target_leg_info.name := none if it is a transparent hw path
    target_leg_info.description := none if it is a transparent hw path

    ret = wsio_reg_legacy(
        NULL,
        &target_leg_info /* pointer to wsio_hw_path_info_t */
    );
    if (ret != WSIO_OK) {
        return;
    }

    /* Parse the list of LUN ids to select the appropriate
     * LUN addressing method. For instance, in pre-11.31 HP-UX,
     * HP-UX FibreChannel I/F drivers applied the following addressing
     * method selection:
     * 1) Logical volume
     * 2) Logical unit
     * 3) Peripheral device
     */
```

1097
Select the new SCSI IO node to which the legacy busses and legacy tgt are going to map (for FC td it is the tp IO node)

For target LUN0 and for every entry in LUN id list {
  \* get LUN id element from lunid_list parameter
  \* If LUN id addressing method does not match the selected addressing method, skip it.
  o) If needed, invoke escsi_get_leg_lun_info() to get the Inquiry data and S/N of the LUN id.
    In case of Tachilte FC, getting LUN0 inquiry data can be useful to determine the type of leg (virtual) bus:
      fcpdev/fcparray.
    \addr.c = iftgt_reg->addr.c;
    \addr.t = iftgt_reg->addr.t;
    \addr.l = lunid_elt.lunid;
    \lpt_info.addr = addr;
    \escsi_get_leg_lun_info(&lpt_info);

2.4 Parse LUN id to identify (virtual) SCSI-2 busses/tgt/luns in a transport specific manner
    and initialize leg_bus_id, leg_tgt_id and leg_lun_id accordingly.
     In pre-11.31, for instance, fcp cdio parses the first level of the LUN id and it uses the following mapping scheme for the Volume Set addressing method:
    7 6 5 4 3 2 1 0
    -----------------
    0 1| bus id
    -----------------
    | tgt id|lun id
    -----------------

    In 11i v3, HP-UX FC drivers extract the leg_bus_id, leg_tgt_id and leg_lun_id information directly from the first level of the SCSI-3 LUN id using the same parsing scheme as in pre-11.31.
  o) For every leg (virtual) bus {
      o) Build leg hw path to (virtual) bus using leg_bus_id
      o) Build legacy virtual bus wsio_legacy_info_t
        bus_leg_info.hw_path = leg_bus_hw_path;
        bus_leg_info.driv_info = legacy I/F driver driv_info
          claiming the legacy hw path
        bus_leg_info.type = T_INTERFACE
        bus_leg_info.id = eg. FC id bytes is a char pointer to a 16 byte array
          byte  0 - bus type
          byte  1 - addressing type
          bytes 8 thru 15 - CRC (S/N
        bus_leg_info.name = transport specific (eg. scsi_c8xx, fcparray, fcpdev)
        bus_leg_info.description = transport specific
      o) Get bus instance number returned by wsio_reg_legacy()
      o) invoke wsio_reg_legacy(&bus_leg_info)
    }
  o) Get bus instance number returned by wsio_reg_legacy()
      and register the legacy bus with the SCSI Services:
      status := escsi_reg_leg_bus(leg bus inst);
      if (status == EINVAL) {
        Invalid arguments were passed. return;
      }
      if (status == EOVERFLOW) {
        We reached the limits of the legacy addressing model. Stop the building of additional legacy hw paths. return;
      }
    o) Get leg bus instance allocated by IOI:
      leg_bus_inst := leg_bus_reg.bus_inst;
    o) For every leg (virtual) target path {
      o) Build legacy (virtual) tgt path hw_path_t
        using leg_bus_hw_path and leg_tgt_id
        leg_tgt_hw_path = hw path to legacy (virtual)
        tgt path
      o) Initialize escsi_reg_leg_tgt_t structure with
appropriate information:
leg_tgt_reg.io_node := new SCSI object IO node
        select in step 3);
leg_tgt_reg.bus_inst := leg_bus_inst;
leg_tgt_reg.tgt_id := leg_tgt_id;
leg_tgt_reg.tgt_hw_path := leg_tgt_hw_path;
o) Register leg (virtual) tgt with SCSI Services
and IOI:
status = escsi_reg_leg_tgt(&leg_tgt_reg);
if (status != ESUCCESS) {
    Invalid arguments were passed or IOI
    registration failed
    return
}
o) For every leg (virtual) lun path {
    -) Build legacy (virtual) LUN path
        hw path from leg tgt path hw path
        and leg_lun_id
        leg_lun_hw_path = hw path to
        leg (virtual) lun path
    -) Build escsi_addr_t corresponding
        to new lpt to LUN id:
        new_lpt_addr.c := iftgt_reg->addr.c
        new_lpt_addr.t := iftgt_reg->addr.t
        new_lpt_addr.l := SCSI-3 LUN id
    -) Register virtual lun path with SCSI
        services and IOI:
        leg_lun_reg.addr := new_lpt_addr;
        leg_lun_reg.bus_inst := leg_bus_inst;
        leg_lun_reg.tgt_id := leg_tgt_id;
        leg_lun_reg.lun_id := leg_lun_id;
        leg_lun_reg.lun_hw_path := leg_lun_hw_path;
        status = escsi_reg_leg_lun(&leg_lun_reg);
        if (status != ESUCCESS) {
            Invalid arguments were passed or IOI
            registration failed
            return
        }
    } /* for every leg (virtual) lun path */
} /* For every leg (virtual) target path */
} /* For every leg (virtual) bus */

4. Return

SEE ALSO

escsi_reg_leg_bus(9F), escsi_reg_leg_tgt(9F), escsi_reg_leg_lun(9F), wsio_reg_legacy(9F)
**NAME**

xxx_if_lpt_close - Allocate and initialize a cdbinfo structure

**SYNOPSIS**

```c
void xxx_if_lpt_close(
    escsi_ifspoc_ws_t *ifspoc
);
```

**PARAMETERS**

*ifspoc* Interface SCSI path open and close structure allocated by SCSI Services.

**DESCRIPTION**

SCSI Services calls the xxx_if_lpt_close entry point to close a lunpath associated with a controller and target path under the interface driver. In the function name, xxx represent the actual interface driver function entry point, which must be specified in the escsi_ifsw_t structure provided by the interface driver.

The *ifspoc* parameter specifies the escsi_addr_t of the lunpath to be closed. The interface driver can pass to the escsi_ifctlr_get, escsi_iftp_get, and escsi_iflpt_get SCSI Services interfaces and get interface-specific data structures associated with this lunpath.

This entry point is asynchronous and SCSI Services provides a callback function in *ifspoc*, which must be called back by interface driver on completion of the lunpath close. The parameters to this callback function are *ifspoc->cbfn.cb_arg* and the errno value indicating the status of lunpath close.

**RETURN VALUE**

The completion status of xxx_if_lpt_close call is passed back to SCSI Services using the callback function provided in escsi_ifspoc_ws_t. The errno values that can be passed back by interface driver as the close completion status in this callback function are:

- **ESUCCESS** The lunpath close was successful.
- **EINVAL** Invalid parameter passed.
- **EBUSY** Transport or device currently busy.
- **ENXIO** Transport connectivity to the specified target path or lunpath failed.
- **EIO** Operation was issued to target and failed at target.
- **ENOMEM** No memory resources to perform requested operation.

**CONSTRAINTS**

Interface drivers must not block in the context of xxx_if_lpt_close.

**EXAMPLE**

None

**SEE ALSO**

1100 SCSI Reference Pages
xxx_if_ioctl(9E)

NAME

xxx_if_ioctl - Interface driver specific ioctl handler

SYNOPSIS

int xxx_if_ioctl(
    dev_t dev,
    escsi_addr_t *addr,
    int cmd,
    caddr_t data,
    int flags
);

PARAMETERS

dev  Device number of device special file on which to issue the ioctl.
addr  The escsi_addr_t of a lunpath.
cmd  The ioctl command.
data  Data associated with the ioctl command.
flags  Flags associated with the ioctl operation.

DESCRIPTION

This entry point is called by SCSI Services to handle:

- An interface driver-specific ioctl that has been issued on a legacy lunpath DSF.
  For instance, pSCSI must support the pSCSI path-specific SIOC ioctls (SIOC_RESET_BUS,
  SIOC_GET_TGT_PARMS, SIOC_GET_BUS_PARMS, SIOC_GET_TGT_LIMITS,
  SIOC_GET_BUS_LIMITS, SIOC_SET_TGT_LIMITS, SIOC_SET_BUS_LIMITS,
  SIOC_RESET_DEV, DIOC_RSTCR) in their ifsw->if_ioctl entry point.
  For instance, FC drivers must support:
    — SIOC_GET_PORT_WWN — Get the target port WWN of the target path corresponding
to a given legacy lunpath DSF.
    — SIOC_GET_WWN — Get the target node WWN of the target path corresponding to a
given legacy lunpath DSF.
- A SCSI Services management operation. The following ioctl is introduced by the SCSI
  Services:
    — SIOC_DEL_IFLPT — This ioctl is invoked by the SCSI Services in the context of an
      lunpath deletion if the interface driver had registered a private object with the lunpath.
      It is defined as follows:
      #define SIOC_DEL_IFLPT               _IO('S', 114)

      It allows the interface driver to delete the interface driver private lunpath object
      associated with the lunpath being deleted. The escsi_addr_t of the lunpath that is
      being deleted is passed as an argument to xxx_if_ioctl. Interface drivers can pass
      escsi_addr_t to escsi_iflpt_get to get their private object associated with lunpath.

RETURN VALUE

ESUCCESS  The ioctl was successful.
EINVAL  Invalid parameter passed.
EBUSY  Transport or device currently busy.
ENXIO Transport connectivity to specified tgt pathname or lunpath failed.
EIO Target returned failure for requested operation.
ENOMEM No memory resources to perform requested operation.
-1 Requested ioctl command not supported.

Other standard HP-UX errno values defined in <sys/errno.h> might also apply.

**CONSTRAINTS**

None

**EXAMPLE**

None

**SEE ALSO**
xxx_if_lpt_open(9E)

NAME

xxx_if_lpt_open - The lunpath open entry point.

SYNOPSIS

xxx_if_lpt_open(
    escsi_ifspoc_ws_t *ifspoc
);

PARAMETERS

ifspoc Interface SCSI path open and close structure allocated by SCSI Services.

DESCRIPTION

SCSI Services calls the xxx_if_lpt_open entry point to open a lunpath associated with a controller and target path under the interface driver. The ifspoc parameter specifies the escsi_addr_t of the lunpath to be opened. Interface driver can pass this to the SCSI Services interfaces escsi_ifctlr_get or escsi_iftp_get and get interface-specific data structures for the controller and tgtpath associated with this lunpath. In the function name, xxx represent the actual interface driver function entry point, which must be specified in the escsi_ifsw_t structure provided by the interface driver.

This entry point is asynchronous and SCSI Services provides a callback function in ifspoc which must be called back by interface driver on completion of the lunpath open. The parameters to this callback function are ifspoc->cbfn.cb_arg and the errno value indicating the status of lunpath open. Interface drivers must not block in the context of xxx_if_lpt_open.

The xxx_if_lpt_open entry point is called only on first open of a lunpath. SCSI Services serializes xxx_if_lpt_open and xxx_if_lpt_close operations for a given lunpath.

During lunpath open, interface driver can allocate an interface-specific data structure for the lunpath (if_lpt) and return it in the escsi_ifspoc_ws_t structure. This is saved by SCSI Services and passed back whenever an I/O is issued to this lunpath.

RETURN VALUE

The completion status of xxx_if_lpt_close call is passed back to SCSI Services using the callback function provided in escsi_ifspoc_ws_t. The errno values that can be passed back by interface driver as the close completion status in this callback function are:

EXAMPLE

None

CONSTRAINTS

None

SEE ALSO

ESUCCESS The lunpath close was successful.
EINVAL Invalid parameter passed.
EBUSY Transport or device currently busy.
ENXIO Transport connectivity to specified tgtpath or lunpath failed.
EIO Operation was issued to target and failed at target.
ENOMEM No memory resources to perform requested operation.
xxx_if_start(9E)

NAME
xxx_if_start - I/O start interface

SYNOPSIS

```c
void xxx_if_start(
    escsi_iobj_t *iobj
);
```

PARAMETERS

iobj Pointer to an I/O object structure.

DESCRIPTION

SCSI Services calls the xxx_if_start entry point to start I/Os queued for an interface driver instance. In the function name, xxx represent the actual interface driver function entry point, which must be specified in the escsi_ifsw_t structure provided by the interface driver.

The iobj parameter is a pointer to the I/O object structure, which contains the select_q associated with the controller or target path on which I/Os need to be started. The select_q is a doubly linked list of scb structures for the I/Os that need to started by the interface driver. The I/O object structure also contains a spinlock and pointer to the if_ctlr interface controller structure. The interface driver must hold the I/O object spinlock while accessing I/Os on the select_q. The select_q is a resource owned by the interface driver, but shared with SCSI Services.

RETURN VALUE

None

CONSTRAINTS

None

EXAMPLE

None

SEE ALSO

ESCSI_IOBJ_LOCK(9G), ESCSI_IOBJ_UNLOCK(9G),
ESCSI_DEQ_FROM_HEAD_FAST(9G), ESCSI_DEQ_FROM_HEAD(9G), escsi_tag_alloc(9F),
escsi_dma_alloc(9F).
xxx_if_task_mgmt(9E)

NAME
xxx_if_task_mgmt - Task management entry point

SYNOPSIS
void xxx_if_task_mgmt(
    escsi_tm_arg_t *tm_arg
);

PARAMETERS
	tm_arg Task management argument structure allocated by SCSI Services.

DESCRIPTION
SCSI Services calls the xxx_if_task_mgmt entry point to issue a task management command to a task set or target under the interface driver. In the function name, xxx represent the actual interface driver function entry point, which must be specified in the escsi_ifsw_t structure provided by the interface driver.

The tm_arg parameter specifies the operation code for the task management command to be issued and provides the buf pointer to be used. From the buf pointer, interface driver can get the scb, which gives links to interface specific data structures associated with the lunpath, target path, and controller.

This entry point is asynchronous and SCSI Services provides a callback function in tm_arg which must be called back by interface driver on completion of the task management operation. The parameters to this callback function are tm_arg and the errno value indicating the task management completion status.

As part of the task management operation, interface driver must abort all pending and active I/Os related to that task set or target path, depending on the scope of the task management operation. This includes I/Os currently on the select_q as well those on interface driver internal queues. The task management callback function provide by SCSI Services must be called back only after all pending and active I/Os for the appropriate scope have been errored back. I/Os must be errored back by calling the I/O completion callback function specified in scb->cbfn.

It is mandatory for interface drivers to support ABORT_TASK_SET task management option. It is recommended that interface drivers support all other task management options, unless there is some transport-specific or HBA-specific limitation in doing so.

RETURN VALUE
The completion status of xxx_if_task_mgmt call is passed back to SCSI Services using the callback function provided in escsi_tm_arg_t. The errno values that can be passed back by interface driver as the task management completion status in the callback function are:

ESUCCESS Task management command successful.
EINVAL Invalid parameter passed.
EBUSY Transport or device currently busy.
ENXIO Transport connectivity to specified tgtpath or lunpath failed.
EIO Operation was issued to target and failed at target.
ENOMEM No memory resources to perform requested operation.

CONSTRAINTS
None
EXAMPLE

None

SEE ALSO
A Deprecated and Obsoleted Interfaces

This appendix contains reference pages that have been deprecated in HP-UX 11i. Developers are encouraged to use the newer interfaces. For information about these interfaces, see the HP-UX 11i v2 Driver Development Reference at:
http://www.hp.com/go/hpux_ddk

<table>
<thead>
<tr>
<th>Deprecated Interface</th>
<th>Description</th>
<th>Replacement Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc_spinlock</td>
<td>Allocate and initialize a spinlock resource.</td>
<td>spin_alloc</td>
</tr>
<tr>
<td>bp_dma_cleanup</td>
<td>Cleanup after a DMA transfer for a list of buffers.</td>
<td>N/A</td>
</tr>
<tr>
<td>bp_dma_setup</td>
<td>Set up a DMA transfer for a list of buffers.</td>
<td>N/A</td>
</tr>
<tr>
<td>b_cpsema</td>
<td>Conditionally acquire (lock) a beta semaphore.</td>
<td>mutex_trylock</td>
</tr>
<tr>
<td>b_initsema</td>
<td>Initialize a beta semaphore.</td>
<td>mutex_init</td>
</tr>
<tr>
<td>b_owns_sema</td>
<td>Test whether a beta semaphore is owned by the calling thread.</td>
<td>mutex_owned</td>
</tr>
<tr>
<td>b_psema</td>
<td>Acquire (lock) a beta semaphore.</td>
<td>mutex_lock</td>
</tr>
<tr>
<td>b_vsema</td>
<td>Release (unlock) a beta semaphore.</td>
<td>mutex_unlock</td>
</tr>
<tr>
<td>cspinlock</td>
<td>Conditionally acquire (lock) a spinlock.</td>
<td>spin_trylock</td>
</tr>
<tr>
<td>dd_close</td>
<td>SCSI driver entry point to handle device close.</td>
<td>N/A</td>
</tr>
<tr>
<td>dd_open</td>
<td>SCSI driver entry point to handle device open.</td>
<td>N/A</td>
</tr>
<tr>
<td>dealloc_spinlock</td>
<td>Deallocate a spinlock resource.</td>
<td>spin_dealloc</td>
</tr>
<tr>
<td>dma_cleanup</td>
<td>Clean up from a DMA transfer.</td>
<td>N/A</td>
</tr>
<tr>
<td>dma_parms</td>
<td>DMA information structure.</td>
<td>N/A</td>
</tr>
<tr>
<td>dma_setup</td>
<td>Set up a DMA transfer.</td>
<td>N/A</td>
</tr>
<tr>
<td>get_sleep_lock</td>
<td>Acquire a sleep queue spinlock.</td>
<td>N/A</td>
</tr>
<tr>
<td>init_map_context</td>
<td>Macro to initialize mapping context structure.</td>
<td>N/A</td>
</tr>
<tr>
<td>iomap_enter_shared_acc</td>
<td>Enter code that accesses shared I/O memory.</td>
<td>user_iomap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user_iomap_private</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel_iomap</td>
</tr>
<tr>
<td>iomap_exit_shared_acc</td>
<td>Exit code that accesses shared I/O memory.</td>
<td>user_iounmap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel_iounmap</td>
</tr>
<tr>
<td>isrlink</td>
<td>Register an interrupt service routine.</td>
<td>wsio_intr_alloc</td>
</tr>
<tr>
<td>isrunlink</td>
<td>Remove the ISR registered by isrlink.</td>
<td>wsio_intr_deactivate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wsio_intr_free</td>
</tr>
<tr>
<td>m_bus_id</td>
<td>Extracts SCSI Bus instance number.</td>
<td>N/A</td>
</tr>
<tr>
<td>m_lun_id</td>
<td>Extracts SCSI LUN ID.</td>
<td>N/A</td>
</tr>
<tr>
<td>m_instance</td>
<td>Get the device instance field from the device number.</td>
<td>wsio_isc_to_instance</td>
</tr>
</tbody>
</table>

Table A-1 Deprecated Interfaces, Their Description, and Replacement
<table>
<thead>
<tr>
<th>Deprecated Interface</th>
<th>Description</th>
<th>Replacement Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_scsi_bus</td>
<td>SCSI bus pointer.</td>
<td>N/A</td>
</tr>
<tr>
<td>m_scsi_isc</td>
<td>Returns isc_table_type pointer.</td>
<td>N/A</td>
</tr>
<tr>
<td>m_scsi_lun</td>
<td>Returns scsi_lun pointer.</td>
<td>N/A</td>
</tr>
<tr>
<td>m_scsi_tgt</td>
<td>Returns scsi_tgt pointer.</td>
<td>N/A</td>
</tr>
<tr>
<td>m_tgt_id</td>
<td>Extracts SCSI target ID.</td>
<td>N/A</td>
</tr>
<tr>
<td>owns_spinlock</td>
<td>Test whether a spinlock is owned by the processor.</td>
<td>spin_owned</td>
</tr>
<tr>
<td>pci_get_fru_info_isc</td>
<td>Get Field Replaceable Unit (FRU) information for the device associated with an ISC.</td>
<td>N/A</td>
</tr>
<tr>
<td>scb</td>
<td>SCSI Control Block structure.</td>
<td>escsi_scb</td>
</tr>
<tr>
<td>scsi_action</td>
<td>Give I/O completion information to SCSI Services.</td>
<td>escsi_action</td>
</tr>
<tr>
<td>scsi_bus_lock</td>
<td>Acquire SCSI bus lock.</td>
<td>N/A</td>
</tr>
<tr>
<td>scsi_bus_unlock</td>
<td>Release SCSI bus lock.</td>
<td>N/A</td>
</tr>
<tr>
<td>scsi_cbfn</td>
<td>SCSI subsystem callback function.</td>
<td>escsi_cbfn</td>
</tr>
<tr>
<td>scsi_Cmd</td>
<td>Prepare driver-generated I/O requests.</td>
<td>N/A</td>
</tr>
<tr>
<td>scsi_cmdx</td>
<td>Prepare driver-generated I/O requests.</td>
<td>escsi_cmdx</td>
</tr>
<tr>
<td>scsi_ddsw</td>
<td>SCSI device switch structure.</td>
<td>escsi_ddsw</td>
</tr>
<tr>
<td>scsi_dequeue</td>
<td>Remove I/O requests from queues maintained by SCSI Services.</td>
<td>N/A</td>
</tr>
<tr>
<td>scsi_dequeue_bp</td>
<td>Remove a specific I/O request from a specified queue maintained by SCSI services.</td>
<td>N/A</td>
</tr>
<tr>
<td>scsi_enqueue</td>
<td>Add buffer bp to a specified queue maintained by SCSI services.</td>
<td>N/A</td>
</tr>
<tr>
<td>scsi_init_inquiry_data</td>
<td>Perform the first inquiry request on a device.</td>
<td>escsi_lpt_open</td>
</tr>
<tr>
<td>scsi_ioctl</td>
<td>Standard SCSI ioctl routine.</td>
<td>escsi_ioctl</td>
</tr>
<tr>
<td>scsi_lun_close</td>
<td>Close a device.</td>
<td>escsi_lun_close</td>
</tr>
<tr>
<td>scsi_lun_open</td>
<td>Open the elements of the hardware path of a SCSI lun.</td>
<td>escsi_lun_open</td>
</tr>
<tr>
<td>scsi_read</td>
<td>Read from device.</td>
<td>escsi_read</td>
</tr>
<tr>
<td>scsi_sense_action</td>
<td>Decode SCSI sense information.</td>
<td>escsi_sense_action</td>
</tr>
<tr>
<td>scsi_strategy</td>
<td>Enqueue the bp to await resources.</td>
<td>escsi_strategy</td>
</tr>
<tr>
<td>scsi_write</td>
<td>Write to device.</td>
<td>escsi_write</td>
</tr>
<tr>
<td>sleep</td>
<td>Sleep on a channel.</td>
<td>cv_wait</td>
</tr>
<tr>
<td>spinlock</td>
<td>Acquire (lock) a spinlock.</td>
<td>spin_lock</td>
</tr>
<tr>
<td>spinunlock</td>
<td>Release (unlock) a spinlock.</td>
<td>spin_unlock</td>
</tr>
<tr>
<td>suser</td>
<td>Test if the current user is a superuser.</td>
<td>priv_policy, priv_scall</td>
</tr>
<tr>
<td>Deprecated Interface</td>
<td>Description</td>
<td>Replacement Interface</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>sw_trigger</td>
<td>Request a software trigger.</td>
<td>N/A</td>
</tr>
<tr>
<td>wakeup</td>
<td>Wake up all threads sleeping on a channel.</td>
<td>cv_signal, cv_broadcast</td>
</tr>
<tr>
<td>wsio_allocate_shared_mem</td>
<td>Allocate and map contiguous memory used for continuous DMA.</td>
<td>wsio_allocate_shared_mem</td>
</tr>
<tr>
<td>wsio_create_attribute</td>
<td>Registers a new attribute with an interface.</td>
<td>prop_create</td>
</tr>
<tr>
<td>wsio_create_interface</td>
<td>Register a new interface with the WSIO.</td>
<td>prop_create</td>
</tr>
<tr>
<td>wsio_destroy_attribute</td>
<td>Destroy an attribute registered with an interface.</td>
<td>prop_destroy</td>
</tr>
<tr>
<td>wsio_destroy_interface</td>
<td>Unregisters an interface with the WSIO.</td>
<td>prop_destroy</td>
</tr>
<tr>
<td>wsio_fastmap</td>
<td>Map all or part of a host address range into an I/O virtual address range.</td>
<td>wsio_fastmap_dma_buffer</td>
</tr>
<tr>
<td>wsio_flush_shared_memory</td>
<td>Flush the memory previously allocated and mapped by wsio_allocate_shared_memory.</td>
<td>wsio_flush_shared_mem</td>
</tr>
<tr>
<td>wsio_free_shared_memory</td>
<td>Release and unmap contiguous memory previously allocated and mapped by wsio_allocate_shared_memory.</td>
<td>wsio_free_shared_mem</td>
</tr>
<tr>
<td>wsio_get_attribute</td>
<td>Gets an attribute registered with an interface.</td>
<td>prop_get</td>
</tr>
<tr>
<td>wsio_get_interrrupts</td>
<td>Determine which interrupt has been assigned to a card.</td>
<td>wsio_get_irq_line</td>
</tr>
<tr>
<td>wsio_get_pva</td>
<td>Translate an I/O virtual address to its processor virtual address.</td>
<td>wsio_iova_to_phys</td>
</tr>
<tr>
<td>wsio_get_registers</td>
<td>Get the register addresses of an interface card.</td>
<td>wsio_get_all_registers</td>
</tr>
<tr>
<td>wsio_get_relationship</td>
<td>Returns the specified relative in the hierarchical tree.</td>
<td>prop_get</td>
</tr>
<tr>
<td>wsio_install_drv_func</td>
<td>Register a driver function with the WSIO driver environment.</td>
<td>wsio_install_drv_event_handler</td>
</tr>
<tr>
<td>wsio_map</td>
<td>Map all or part of a host address range into an I/O virtual address range.</td>
<td>wsio_map_dma_buffer</td>
</tr>
<tr>
<td>wsio_modify_attribute</td>
<td>Modifies an existing attribute.</td>
<td>prop_modify</td>
</tr>
<tr>
<td>wsio_register_probe_func</td>
<td>Insert a driver-specified probe function into the global probe list.</td>
<td>wsio_register_dev_probe, wsio_register_addr_probe</td>
</tr>
<tr>
<td>wsio_remap</td>
<td>Map a host range into a pre-mapped I/O Virtual Address range.</td>
<td>wsio_remap_dma_buffer</td>
</tr>
<tr>
<td>wsio_set_attributes</td>
<td>Set map function attributes.</td>
<td>wsio_dma_set_device_attributes, wsio_set_dma_attributes</td>
</tr>
<tr>
<td>wsio_sizeof_attribute</td>
<td>Returns the size of an attribute.</td>
<td>prop_size</td>
</tr>
</tbody>
</table>