
HPSS

Programmer's Reference Guide, Volume 2

**High Performance Storage System
Release 4.1.1**

April 1999 (Revision 0)

HPSS Programmer's Reference, Volume 2

Copyright (C) 1992-1999 International Business Machines Corporation, The Regents of the University of California, Sandia Corporation, Lockheed Martin Energy Research Corporation, and NASA Langley Research Center.

All rights reserved.

Portions of this work were produced by the University of California, Lawrence Livermore National Laboratory (LLNL) under Contract No. W-7405-ENG-48 with the U.S. Department of Energy (DOE), by the University of California, Lawrence Berkeley National Laboratory (LBNL) under Contract No. DEAC03776SF00098 with DOE, by the University of California, Los Alamos National Laboratory (LANL) under Contract No. W-7405-ENG-36 with DOE, by Sandia Corporation, Sandia National Laboratories (SNL) under Contract No. DEAC0494AL85000 with DOE, and Lockheed Martin Energy Research Corporation, Oak Ridge National Laboratory (ORNL) under Contract No. DE-AC05-96OR22464 with DOE. The U.S. Government has certain reserved rights under its prime contracts with the Laboratories.

DISCLAIMER

Portions of this software were sponsored by an agency of the United States Government. Neither the United States, DOE, The Regents of the University of California, Sandia Corporation, Lockheed Martin Energy Research Corporation, nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

Printed in the United States of America

HPSS Release 4.1.1

April 1999 (Revision 0)

High Performance Storage System is a registered trademark of International Business Machines Corporation.
IBM is a registered trademark of International Business Machines Corporation.
AIX and RISC/6000 are trademarks of International Business Machines Corporation.
Encina is a registered trademark of Transarc Corporation.
UNIX is a registered trademark of UNIX System Laboratories, Inc.
Sammi is a trademark of Scientific Software Intercomp.
NFS and Network File System are trademarks of Sun Microsystems, Inc.
DST is a trademark of Ampex Systems Corporation.
ACLS is a trademark of Storage Technology Corporation.
Other brands and product names appearing herein may be trademarks or registered trademarks of third parties.

| | | |
|--------------------------|----------------|------------|
| Table of Contents | Preface | xii |
|--------------------------|----------------|------------|

| | | |
|----------------------------------|--|------------|
| 1. Overview | | 1-1 |
| 1.1. Name Server | | 1-1 |
| 1.1.1. Purpose | | 1-1 |
| 1.1.2. Components | | 1-1 |
| 1.1.3. Constraints | | 1-1 |
| 1.1.4. Libraries | | 1-1 |
| 1.2. Bitfile Server | | 1-2 |
| 1.2.1. Purpose | | 1-2 |
| 1.2.2. Bitfile Server Components | | 1-2 |
| 1.2.3. Constraints | | 1-2 |
| 1.2.4. Libraries | | 1-3 |
| 1.3. Storage Server | | 1-3 |
| 1.3.1. Purpose | | 1-3 |
| 1.3.2. Components | | 1-3 |
| 1.3.3. Constraints | | 1-4 |
| 1.3.4. Libraries | | 1-4 |
| 1.4. Mover | | 1-4 |
| 1.4.1. Purpose | | 1-4 |
| 1.4.2. Components | | 1-5 |
| 1.4.3. Constraints | | 1-5 |
| 1.4.4. Libraries | | 1-6 |
| 1.5. Physical Volume Library | | 1-6 |
| 1.5.1. Purpose | | 1-6 |
| 1.5.2. Components | | 1-6 |
| 1.5.3. Constraints | | 1-7 |
| 1.5.4. Libraries | | 1-7 |
| 1.6. Physical Volume Repository | | 1-7 |
| 1.6.1. Purpose | | 1-7 |
| 1.6.2. Components | | 1-7 |
| 1.6.3. Constraints | | 1-8 |
| 1.6.4. Libraries | | 1-8 |
| 1.7. System Manager | | 1-9 |
| 1.7.1. Purpose | | 1-9 |
| 1.7.2. Components | | 1-9 |
| 1.7.3. Constraints | | 1-14 |
| 1.7.4. Libraries | | 1-15 |
| 1.8. Location Server | | 1-15 |
| 1.8.1. Purpose | | 1-15 |
| 1.8.2. Components | | 1-15 |
| 1.8.3. Constraints | | 1-15 |
| 1.8.4. Libraries | | 1-16 |
| 1.8.5. Referenced Data Types | | 1-16 |

| | | |
|-----------|---|------------|
| 2. | Name Server Functions..... | 2-1 |
| 2.1. | API Functions..... | 2-1 |
| 2.1.1. | ns_Delete | 2-2 |
| 2.1.2. | ns_DeleteACL | 2-5 |
| 2.1.3. | ns_DeleteFileset..... | 2-8 |
| 2.1.4. | ns_GetACL | 2-10 |
| 2.1.5. | ns_GetAttrs | 2-12 |
| 2.1.6. | ns_GetFilesetAttrs..... | 2-15 |
| 2.1.7. | ns_GetFilesetByNameOrId | 2-17 |
| 2.1.8. | ns_GetName | 2-19 |
| 2.1.9. | ns_Insert..... | 2-21 |
| 2.1.10. | ns_Mkdir | 2-25 |
| 2.1.11. | ns_MkFileset | 2-28 |
| 2.1.12. | ns_MkJunction | 2-31 |
| 2.1.13. | ns_MkLink | 2-34 |
| 2.1.14. | ns_MkSymLink | 2-37 |
| 2.1.15. | ns_NSGetAttrs | 2-40 |
| 2.1.16. | ns_NSSetAttrs | 2-42 |
| 2.1.17. | ns_ReadDir..... | 2-44 |
| 2.1.18. | ns_ReadFilesetAttrs | 2-47 |
| 2.1.19. | ns_ReadGlobalFilesets | 2-49 |
| 2.1.20. | ns_ReadJunctionPathNames..... | 2-52 |
| 2.1.21. | ns_ReadLink | 2-54 |
| 2.1.22. | ns_Rename | 2-56 |
| 2.1.23. | ns_ServerGetAttrs..... | 2-59 |
| 2.1.24. | ns_ServerSetAttrs | 2-61 |
| 2.1.25. | ns_SetACL | 2-63 |
| 2.1.26. | ns_SetAttrs..... | 2-65 |
| 2.1.27. | ns_SetFilesetAttrs | 2-68 |
| 2.1.28. | ns_Statistics | 2-70 |
| 2.1.29. | ns_UpdateACL | 2-72 |
| 2.2. | Data Definitions..... | 2-75 |
| 2.2.1. | Access Control List Conformant Array - ns_ACLConfArray_t..... | 2-75 |
| 2.2.2. | Access Control List Entry - ns_ACLEntry_t..... | 2-75 |
| 2.2.3. | Attribute Bit Map - ns_AttrBits_t | 2-76 |
| 2.2.4. | Name Server Directory Entry - ns_DirEntry_t..... | 2-80 |
| 2.2.5. | Name Server Fileset Bit Map - ns_FilesetAttrBits_t..... | 2-81 |
| 2.2.6. | Name Server Fileset Attrs structure - ns_FilesetAttrs_t..... | 2-82 |
| 2.2.7. | Name Server FilesetAttrs Conformant Array - ns_FilesetAttrsConfArray_t..... | 2-83 |
| 2.2.8. | Name Server FilesetAttrs Entry - ns_FilesetAttrsEntry_t..... | 2-84 |
| 2.2.9. | Name Server Global Fileset Conformant Array - ns_GFilesetConfArray_t | 2-85 |
| 2.2.10. | Name Server GlobalFileset Entry - ns_GlobalFilesetEntry_t..... | 2-85 |
| 2.2.11. | Name Server Junction Path Conformant Array - ns_JunctionPathConfArray_t..... | 2-86 |
| 2.2.12. | Name Server Junction Path Entry - ns_JunctionPathEntry_t..... | 2-87 |

| | | |
|-----------|--|------------|
| 2.2.13. | Name Server Object Handle - ns_ObjHandle_t | 2-87 |
| 2.2.14. | Name Server Return Structure - ns_RemainingPath_t..... | 2-88 |
| 2.2.15. | Name Server Configuration - ns_SpecificConfig_t..... | 2-89 |
| 2.2.16. | Name Server Statistics Structure - ns_StatisticsRec_t | 2-92 |
| 3. | Bitfile Server Functions | 3-1 |
| 3.1. | API Functions..... | 3-1 |
| 3.1.1. | bfs_BitfileGetAttrs..... | 3-2 |
| 3.1.2. | bfs_BitfileGetXAttrs | 3-4 |
| 3.1.3. | bfs_BitfileOpenGetAttrs..... | 3-6 |
| 3.1.4. | bfs_BitfileOpenSetAttrs | 3-8 |
| 3.1.5. | bfs_BitfileSetAttrs | 3-10 |
| 3.1.6. | bfs_BitfileOpenSetCosByHints | 3-12 |
| 3.1.7. | bfs_Clear | 3-14 |
| 3.1.8. | bfs_Close | 3-16 |
| 3.1.9. | bfs_CopyFile..... | 3-18 |
| 3.1.10. | bfs_Create..... | 3-20 |
| 3.1.11. | bfs_GetCOSStats..... | 3-22 |
| 3.1.12. | bfs_Migrate..... | 3-24 |
| 3.1.13. | bfs_Open..... | 3-26 |
| 3.1.14. | bfs_Purge | 3-29 |
| 3.1.15. | bfs_Read | 3-31 |
| 3.1.16. | bfs_ServerGetAttrs..... | 3-33 |
| 3.1.17. | bfs_ServerSetAttrs | 3-34 |
| 3.1.18. | bfs_Stage | 3-36 |
| 3.1.19. | bfs_StageCallBack | 3-38 |
| 3.1.20. | bfs_Unlink..... | 3-40 |
| 3.1.21. | bfs_Write | 3-42 |
| 3.1.22. | Bitfile Volatile and Metadata Attributes - bf_attrib_t..... | 3-44 |
| 3.1.23. | Bitfile Metadata Attributes - bf_attrib_md_t..... | 3-45 |
| 3.1.24. | Bitfile Descriptor - bf_descriptor_md_t..... | 3-47 |
| 3.1.25. | Bitfile Storage Level Statistics - bf_level_stats_md..... | 3-48 |
| 3.1.26. | Bitfile Managed Object Data Structure - bfMO_attrib_t..... | 3-49 |
| 3.1.27. | Bitfile Open Context - bf_open_context..... | 3-49 |
| 3.1.28. | Bitfile Open Context List - bf_open_context_list_t..... | 3-51 |
| 3.1.29. | Bitfile Open Context Header - bf_open_context_hdr_t..... | 3-51 |
| 3.1.30. | Bitfile Tape Segment Metadata - bf_tape_segment_md_t..... | 3-52 |
| 3.1.31. | Bitfile Disk Segment Metadata - bf_disk_segment_md_t | 3-53 |
| 3.1.32. | Bitfile Disk Segment Region - bf_disk_region_md_t..... | 3-54 |
| 3.1.33. | Bitfile Disk Allocation Map Metadata - bf_disk_alloc_rec_md_t..... | 3-55 |
| 3.1.34. | Class of Service - hpss_cos_md_t..... | 3-56 |
| 3.1.35. | Class of Service Hints - hpss_cos_hints_t..... | 3-58 |
| 3.1.36. | Class of Service Priorities - hpss_cos_priorities_t | 3-59 |
| 3.1.37. | Owner Record - bfs_owner_rec_t | 3-61 |
| 3.1.38. | Request Attributes - req_attrib_t | 3-61 |
| 3.1.39. | Reverse Map Field - rev_map_t..... | 3-62 |

| | | |
|-----------|--|------------|
| 3.1.40. | Bitfile Cache Entry - bf_cache_entry_t | 3-63 |
| 3.1.41. | Bitfile Cache Hash - bf_cache_hash_t | 3-64 |
| 3.1.42. | Bitfile Segments Cache Entry - bf_segments_cache_entry_t | 3-65 |
| 3.1.43. | Storage Segment Delete Entry - sseg_delete_entry_t | 3-66 |
| 3.1.44. | Current Bitfile Segment Information - current_segment_info_t | 3-66 |
| 3.1.45. | Bitfile Disk Map - bf_disk_map_t | 3-67 |
| 3.1.46. | Bitfile Server Connect Context - bfs_connect_context_t | 3-68 |
| 3.1.47. | HPSS Segment List - hpss_segment_list_t | 3-69 |
| 3.1.48. | HPSS Segment Descriptor - hpss_segment_desc_t | 3-69 |
| 3.1.49. | HPSS Background Stage CallBack Structure - bfs_callback_addr_t3-70 | |
| 3.2. | Other Interfaces (OFD and Request list) | 3-70 |
| 3.2.1. | hpss_InitOfdMgr | 3-71 |
| 3.2.2. | hpss_GetOfd | 3-72 |
| 3.2.3. | hpss_FreeOfd..... | 3-74 |
| 3.2.4. | hpss_CloseAllOfds | 3-75 |
| 3.2.5. | hpss_CleanupOfds..... | 3-76 |
| 3.2.6. | hpss_InitOfdList..... | 3-77 |
| 3.2.7. | hpss_ReqListDeleteEntry | 3-78 |
| 3.2.8. | hpss_ReqListFindReqld | 3-79 |
| 3.2.9. | hpss_ReqListInit | 3-80 |
| 3.2.10. | hpss_ReqListInsertEntry | 3-81 |
| 3.2.11. | hpss_ReqListNextEntry | 3-82 |
| 3.2.12. | hpss_ReqListSetState | 3-83 |
| 3.3. | Other Data Definitions (OFD and request list) | 3-84 |
| 3.3.1. | HPSS Open File Descriptor (OFD) - hpss_ofd_t..... | 3-84 |
| 3.3.2. | HPSS Open File Descriptor List Header - hpss_ofd_hdr_t | 3-85 |
| 3.3.3. | Request List - hpss_reqlist_t | 3-86 |
| 3.3.4. | Request List Entry - hpss_reqlist_entry_t..... | 3-86 |
| 4. | Storage Server Functions | 4-1 |
| 4.1. | API Functions..... | 4-1 |
| 4.1.1. | ss_BeginSession | 4-2 |
| 4.1.2. | ss_EndSession..... | 4-4 |
| 4.1.3. | ss_GetStorageClassStats | 4-6 |
| 4.1.4. | ss_GetWaitingEvents | 4-8 |
| 4.1.5. | ss_MapCreate | 4-10 |
| 4.1.6. | ss_MapDelete..... | 4-12 |
| 4.1.7. | ss_MapGetAttrs..... | 4-13 |
| 4.1.8. | ss_MapSetAttrs | 4-14 |
| 4.1.9. | ss_PVCreate | 4-16 |
| 4.1.10. | ss_PVDelete..... | 4-18 |
| 4.1.11. | ss_PVGetAttrs | 4-19 |
| 4.1.12. | ss_PVMount | 4-21 |
| 4.1.13. | ss_PVRead..... | 4-23 |
| 4.1.14. | ss_PVSetAttrs | 4-25 |
| 4.1.15. | ss_PVUnmount | 4-27 |

| | | |
|---------|---|-------|
| 4.1.16. | ss_PVWrite..... | 4-28 |
| 4.1.17. | ss_ServerGetAttrs | 4-30 |
| 4.1.18. | ss_ServerSetAttrs | 4-31 |
| 4.1.19. | ss_SSCopySegment | 4-33 |
| 4.1.20. | ss_SSCreate | 4-35 |
| 4.1.21. | ss_SSDelete..... | 4-38 |
| 4.1.22. | ss_SSDeleteList | 4-40 |
| 4.1.23. | ss_SSGetAttrs | 4-42 |
| 4.1.24. | ss_SSMount | 4-43 |
| 4.1.25. | ss_SSMoveSegment..... | 4-45 |
| 4.1.26. | ss_SSRead..... | 4-47 |
| 4.1.27. | ss_SSRvGetAttrs | 4-49 |
| 4.1.28. | ss_SSRvSetAttrs | 4-50 |
| 4.1.29. | ss_SSSetAttrs | 4-52 |
| 4.1.30. | ss_SSStartMount..... | 4-54 |
| 4.1.31. | ss_SSUnlink | 4-56 |
| 4.1.32. | ss_SSUnmount | 4-58 |
| 4.1.33. | ss_SSWrite..... | 4-59 |
| 4.1.34. | ss_VVCreate | 4-61 |
| 4.1.35. | ss_VVDelete..... | 4-63 |
| 4.1.36. | ss_VVGetAttrs | 4-65 |
| 4.1.37. | ss_VVMount | 4-67 |
| 4.1.38. | ss_VVRead..... | 4-69 |
| 4.1.39. | ss_VVSetAttrs | 4-71 |
| 4.1.40. | ss_VVUnmount | 4-73 |
| 4.1.41. | ss_VVWrite..... | 4-74 |
| 4.2. | Data Definitions..... | 4-76 |
| 4.2.1. | Storage Server Attribute Record - ssrv_attr_t | 4-76 |
| 4.2.2. | Storage Segment Record - storage_segment_record_t | 4-77 |
| 4.2.3. | Storage Segment Attribute Record - ss_attr_t | 4-79 |
| 4.2.4. | Storage Segment Metadata - storage_segment_md_t | 4-80 |
| 4.2.5. | Storage Map Record - storage_map_record_t..... | 4-82 |
| 4.2.6. | Storage Map Attribute Record - ss_map_attr_t..... | 4-84 |
| 4.2.7. | Tape Storage Map Metadata - storage_map_md_t | 4-84 |
| 4.2.8. | Disk Storage Map Metadata..... | 4-87 |
| 4.2.9. | Virtual Volume Record - virtual_volume_record_t..... | 4-88 |
| 4.2.10. | Virtual Volume Attribute Record - vv_attr_t..... | 4-91 |
| 4.2.11. | Virtual Volume Metadata - virtual_volume_md_t..... | 4-92 |
| 4.2.12. | Physical Volume Record - physical_volume_record_t | 4-95 |
| 4.2.13. | Physical Volume Attribute Record - pv_attr_t..... | 4-98 |
| 4.2.14. | Physical Volume Metadata - physical_volume_md_t | 4-99 |
| 4.2.15. | Device Table Record - device_table_record_t | 4-102 |
| 4.2.16. | Session Record - ss_session_t..... | 4-103 |
| 4.2.17. | Relative Address - relative_address_t..... | 4-105 |
| 4.2.18. | Composite Address - composite_address_t | 4-105 |
| 4.2.19. | Absolute Address - absolute_address_t..... | 4-106 |
| 4.2.20. | Physical Volume List - pv_list_t | 4-106 |

| | | |
|-----------|---|------------|
| 4.2.21. | Physical Volume List Element - pv_list_element_t..... | 4-106 |
| 4.2.22. | Owner Record - owner_rec_t..... | 4-107 |
| 4.2.23. | Wait List - waitlist_t..... | 4-107 |
| 4.2.24. | Storage Class Array - ss_class_array_t..... | 4-108 |
| 4.2.25. | Storage Class Array Element - ss_class_t..... | 4-108 |
| 4.2.26. | Event Array - ss_sclass_array_t..... | 4-109 |
| 4.2.27. | Event Array Element - ss_event_rec_t..... | 4-109 |
| 4.2.28. | Segment Array..... | 4-110 |
| 4.2.29. | Delete segment array..... | 4-110 |
| 4.2.30. | Delete Segment Array Element..... | 4-110 |
| 4.2.31. | Copy Control Block..... | 4-111 |
| 5. | Mover Functions..... | 5-1 |
| 5.1. | API Functions..... | 5-1 |
| 5.1.1. | mvr_Abort..... | 5-2 |
| 5.1.2. | mvr_CreateDevice..... | 5-3 |
| 5.1.3. | mvr_DeleteDevice..... | 5-5 |
| 5.1.4. | mvr_DeviceGetAttrs..... | 5-7 |
| 5.1.5. | mvr_DeviceGetAttrs_IOD..... | 5-9 |
| 5.1.6. | mvr_DeviceSetAttrs..... | 5-10 |
| 5.1.7. | mvr_DeviceSetAttrs_IOD..... | 5-12 |
| 5.1.8. | mvr_DeviceSpec..... | 5-14 |
| 5.1.9. | mvr_MVRGetAttrs..... | 5-16 |
| 5.1.10. | mvr_MVRSetAttrs..... | 5-17 |
| 5.1.11. | mvr_Read..... | 5-19 |
| 5.1.12. | mvr_ServerGetAttrs..... | 5-21 |
| 5.1.13. | mvr_ServerSetAttrs..... | 5-22 |
| 5.1.14. | mvr_Write..... | 5-24 |
| 5.2. | Data Definitions..... | 5-26 |
| 5.2.1. | Mover State Structure - mover_attr_t..... | 5-26 |
| 5.2.2. | Device Descriptor - devdesc_attr_t..... | 5-27 |
| 5.2.3. | Mover Configuration Structure - mvr_config_t..... | 5-32 |
| 5.2.4. | Mover Protocol Message Structures..... | 5-33 |
| 6. | Physical Volume Library Functions..... | 6-1 |
| 6.1. | API Functions..... | 6-1 |
| 6.1.1. | pvl_AllocateVol..... | 6-2 |
| 6.1.2. | pvl_CancelAllJobs..... | 6-4 |
| 6.1.3. | pvl_CreateDrive..... | 6-5 |
| 6.1.4. | pvl_DeallocateVol..... | 6-6 |
| 6.1.5. | pvl_DeleteDrive..... | 6-7 |
| 6.1.6. | pvl_DismountDrive..... | 6-8 |
| 6.1.7. | pvl_DismountJobId..... | 6-9 |
| 6.1.8. | pvl_DismountVolume..... | 6-10 |
| 6.1.9. | pvl_DriveGetAttrs..... | 6-11 |
| 6.1.10. | pvl_DriveSetAttrs..... | 6-12 |

| | | |
|-----------|---|------------|
| 6.1.11. | pvl_Export..... | 6-14 |
| 6.1.12. | pvl_Import..... | 6-15 |
| 6.1.13. | pvl_Mount..... | 6-17 |
| 6.1.14. | pvl_MountAdd..... | 6-19 |
| 6.1.15. | pvl_MountCommit | 6-21 |
| 6.1.16. | pvl_MountCompleted..... | 6-23 |
| 6.1.17. | pvl_MountNew | 6-25 |
| 6.1.18. | pvl_Move | 6-27 |
| 6.1.19. | pvl_NotifyCartridge | 6-29 |
| 6.1.20. | pvl_PVLGetAttrs | 6-31 |
| 6.1.21. | pvl_PVLSetAttrs | 6-32 |
| 6.1.22. | pvl_QueueGetAttrs | 6-34 |
| 6.1.23. | pvl_QueueSetAttrs | 6-35 |
| 6.1.24. | pvl_RequestGetAttrs | 6-37 |
| 6.1.25. | pvl_RequestSetAttrs..... | 6-38 |
| 6.1.26. | pvl_ServerGetAttrs | 6-40 |
| 6.1.27. | pvl_ServerSetAttrs..... | 6-41 |
| 6.1.28. | pvl_VolumeGetAttrs | 6-43 |
| 6.1.29. | pvl_VolumeSetAttrs..... | 6-44 |
| 6.1.30. | pvl_WriteVolumeLabel | 6-46 |
| 6.2. | Data Definitions..... | 6-47 |
| 6.2.1. | PVL Data Structure - pvl_data_t..... | 6-47 |
| 6.2.2. | Queue Data Structure - api_queue_data_t | 6-48 |
| 6.2.3. | PVL Job Queue Entry - request_data_t..... | 6-50 |
| 6.2.4. | Cartridge ID Structure - cart_t | 6-53 |
| 6.2.5. | Volume Structure - vol_t..... | 6-54 |
| 6.2.6. | Media Type Structure - media_type_t | 6-54 |
| 6.2.7. | Active Volume State Structure - activity_data_t | 6-55 |
| 6.2.8. | Activity Structure - activity_t..... | 6-57 |
| 6.2.9. | Client Information Structure - client_info_t | 6-58 |
| 6.2.10. | Job Data Structure - job_data_t | 6-59 |
| 6.2.11. | Job Entry Structure - job_ent_t..... | 6-61 |
| 6.2.12. | Cartridge List Entry Structure - cart_ent_t..... | 6-62 |
| 6.2.13. | Volume Data Structure - vol_data_t | 6-62 |
| 6.2.14. | Drive Data Structure - drive_data_t..... | 6-64 |
| 6.2.15. | Drive Index - drive_index_t..... | 6-65 |
| 6.2.16. | Drive ID - drive_t..... | 6-65 |
| 6.2.17. | Drive Type - drive_type_t..... | 6-66 |
| 6.2.18. | Drive Type Entry - drive_type_ent_t | 6-66 |
| 6.2.19. | Job ID - job_id_t | 6-67 |
| 6.2.20. | PVR Index - pvr_index_t..... | 6-67 |
| 6.2.21. | Queue Data - queue_data_t..... | 6-67 |
| 6.3. | Other Interfaces | 6-69 |
| 6.3.1. | ss_MountCallback | 6-69 |
| 7. | Physical Volume Repository Functions | 7-1 |

| | | |
|-----------|---|------------|
| 7.1. | API Functions..... | 7-1 |
| 7.1.1. | pvr_Audit | 7-2 |
| 7.1.2. | pvr_CartridgeGetAttrs | 7-4 |
| 7.1.3. | pvr_CartridgeSetAttrs | 7-5 |
| 7.1.4. | pvr_CheckIn | 7-7 |
| 7.1.5. | pvr_CheckOut | 7-8 |
| 7.1.6. | pvr_DismountCart | 7-10 |
| 7.1.7. | pvr_DismountDrive | 7-11 |
| 7.1.8. | pvr_Eject..... | 7-12 |
| 7.1.9. | pvr_Inject..... | 7-14 |
| 7.1.10. | pvr_ListAllCart | 7-15 |
| 7.1.11. | pvr_ListPendingMounts | 7-17 |
| 7.1.12. | pvr_Mount..... | 7-19 |
| 7.1.13. | pvr_MountComplete | 7-21 |
| 7.1.14. | pvr_PVRGetAttrs | 7-23 |
| 7.1.15. | pvr_PVRSetAttrs | 7-24 |
| 7.1.16. | pvr_ServerGetAttrs..... | 7-26 |
| 7.1.17. | pvr_ServerSetAttrs | 7-27 |
| 7.2. | Device Interfaces | 7-29 |
| 7.2.1. | device_Audit | 7-29 |
| 7.2.2. | device_Dismount..... | 7-30 |
| 7.2.3. | device_Eject | 7-32 |
| 7.2.4. | device_Init | 7-33 |
| 7.2.5. | device_Inject..... | 7-34 |
| 7.2.6. | device_LocationToString | 7-35 |
| 7.2.7. | device_Mount | 7-36 |
| 7.2.8. | device_MountComplete..... | 7-38 |
| 7.2.9. | device_Release | 7-39 |
| 7.2.10. | device_SetDrive | 7-40 |
| 7.3. | Data Definitions..... | 7-41 |
| 7.3.1. | Cartridge Side - side_t..... | 7-41 |
| 7.3.2. | drive_addr_t..... | 7-41 |
| 7.3.3. | ioport_addr_t | 7-41 |
| 7.3.3. | location_t | 7-42 |
| 7.3.4. | cart_data_t..... | 7-42 |
| 7.3.5. | pvr_data_t..... | 7-45 |
| 7.3.6. | Manufacturing Lot Number - lot_number_t | 7-46 |
| 7.3.7. | Cartridge Manufacturer - manufacturer_t..... | 7-46 |
| 7.3.8. | Check-in request - checkin_req_t | 7-46 |
| 7.3.9. | Other APIs..... | 7-48 |
| 7.3.9.1. | pvl_MountCompleted | 7-48 |
| 8. | System Manager Functions | 8-1 |
| 8.1. | API Functions..... | 8-1 |
| 8.1.1. | ssm_AcctChange | 8-2 |
| 8.1.2. | ssm_AcctRun | 8-4 |

| | | |
|---------|---|------|
| 8.1.3. | ssm_Adm | 8-5 |
| 8.1.4. | ssm_AttrGet | 8-8 |
| 8.1.5. | ssm_AttrReg..... | 8-10 |
| 8.1.6. | ssm_AttrSet..... | 8-12 |
| 8.1.7. | ssm_CartExport..... | 8-14 |
| 8.1.8. | ssm_CartImport..... | 8-16 |
| 8.1.9. | ssm_CartMove | 8-18 |
| 8.1.10. | ssm_CheckIn..... | 8-20 |
| 8.1.11. | ssm_CheckOut..... | 8-22 |
| 8.1.12. | ssm_ConfigAdd | 8-24 |
| 8.1.13. | ssm_ConfigDelete | 8-26 |
| 8.1.14. | ssm_ConfigGetDefault | 8-28 |
| 8.1.15. | ssm_ConfigRead..... | 8-31 |
| 8.1.16. | ssm_ConfigUpdate..... | 8-33 |
| 8.1.17. | ssm_Delog | 8-35 |
| 8.1.18. | ssm_DriveDismount | 8-37 |
| 8.1.19. | ssm_FilesetCreate | 8-39 |
| 8.1.20. | ssm_FilesetDelete..... | 8-41 |
| 8.1.21. | ssm_JobCancel..... | 8-43 |
| 8.1.22. | ssm_JunctionCreate..... | 8-45 |
| 8.1.23. | ssm_JunctionDelete | 8-47 |
| 8.1.24. | ssm_ResourceCreate..... | 8-49 |
| 8.1.25. | ssm_ResourceDelete | 8-51 |
| 8.1.26. | ssm_ResourceReclaim | 8-53 |
| 8.1.27. | ssm_ResourceRepack | 8-55 |
| 8.2. | APIs Available to the Other HPSS Subsystems | 8-57 |
| 8.2.1. | ssm_BitfileNotify | 8-58 |
| 8.2.2. | ssm_CartNotify | 8-60 |
| 8.2.3. | ssm_DeviceNotify..... | 8-62 |
| 8.2.4. | ssm_DMGFilesetNotify..... | 8-64 |
| 8.2.5. | ssm_DMGNotify | 8-66 |
| 8.2.6. | ssm_DriveNotify | 8-68 |
| 8.2.7. | ssm_LogFileNotify..... | 8-70 |
| 8.2.8. | ssm_LogMsgNotify..... | 8-72 |
| 8.2.9. | ssm_LSStatsNotify | 8-73 |
| 8.2.10. | ssm_MPSNotify | 8-75 |
| 8.2.11. | ssm_MPS_SClassNotify | 8-77 |
| 8.2.12. | ssm_MVRNotify..... | 8-79 |
| 8.2.13. | ssm_MapNotify..... | 8-81 |
| 8.2.14. | ssm_MountNotify..... | 8-83 |
| 8.2.15. | ssm_NFS2_StatsNotify | 8-85 |
| 8.2.16. | ssm_NSFilesetNotify | 8-87 |
| 8.2.17. | ssm_NSNotify..... | 8-89 |
| 8.2.18. | ssm_PVLNotify | 8-91 |
| 8.2.19. | ssm_PVNotify | 8-93 |
| 8.2.20. | ssm_PVRNotify | 8-95 |
| 8.2.21. | ssm_QueueNotify..... | 8-97 |

| | | |
|-----------|---|-------|
| 8.2.22. | ssm_RequestNotify | 8-99 |
| 8.2.23. | ssm_SFSNotify..... | 8-101 |
| 8.2.24. | ssm_SSNotify..... | 8-103 |
| 8.2.25. | ssm_ServerNotify..... | 8-105 |
| 8.2.26. | ssm_SsrvNotify | 8-107 |
| 8.2.27. | ssm_TapeCheckInNotify | 8-109 |
| 8.2.27. | ssm_VVNotify..... | 8-111 |
| 8.2.28. | ssm_VolNotify..... | 8-113 |
| 8.3. | Data Definitions..... | 8-115 |
| 8.3.1. | Data Common to the System Manager and the Data Server..... | 8-115 |
| 8.3.1.1. | Significant constants | 8-115 |
| 8.3.1.2. | Server List - ServerList_t..... | 8-115 |
| 8.3.1.3. | Drive List - DriveList_t | 8-119 |
| 8.3.1.4. | Class of Service List - COSList_t..... | 8-120 |
| 8.3.1.5. | Storage Class List - SClassList_t..... | 8-121 |
| 8.3.1.6. | Hierarchy List - HierList_t | 8-122 |
| 8.3.1.7. | Migration Policy List - MigrPList_t | 8-123 |
| 8.3.1.8. | Purge Policy List - PurgPList_t..... | 8-123 |
| 8.3.1.9. | Notification Structure - NotifyUnion_t..... | 8-124 |
| 8.3.1.10. | Server Info Data Structure - SrvInfoUnion_t | 8-124 |
| 8.3.1.11. | Drive Data ID - DriveDataID_t..... | 8-132 |
| 8.3.1.12. | Drive Data for Configuration Operations - DriveData_t..... | 8-133 |
| 8.3.1.13. | Cartridge Import Data - PvlImport_t | 8-133 |
| 8.3.1.14. | Storage Server Resource Data Structure - SsResources_t.... | 8-135 |
| 8.3.1.15. | Storage Server Repack Structure - SsRepack_t..... | 8-135 |
| 8.3.1.16. | Storage Server Reclaim Structure - SsReclaim_t..... | 8-136 |
| 8.3.1.17. | ClientID..... | 8-137 |
| 8.3.2. | Data Private to the System Manager..... | 8-137 |
| 8.3.2.1. | Table of Registered Clients - client_list_t..... | 8-137 |
| 8.3.2.2. | Server Network Connection Table - server_net_t..... | 8-139 |
| 8.3.2.3. | Table of Registered Managed Object Attributes - registered_mo_t..... | 8-141 |
| 8.3.2.4. | Notification Queues..... | 8-142 |
| 8.3.2.5. | Configuration File List - config_file_list_t..... | 8-144 |
| 8.3.2.6. | Copy of the HPSS Server Configuration File - server_config_list_t..... | 8-145 |
| 8.3.2.7. | Condition Variable Structure - condition_variable_t | 8-146 |
| 8.3.2.8. | Drive Data Structure - DriveDataID_t..... | 8-146 |
| 8.3.2.9. | Bitfile ID Register Structure - ssm_bitfile_reg_id_t..... | 8-146 |
| 8.3.2.10. | Descriptive Name - ssm_descname_t | 8-147 |
| 8.3.2.11. | Bitfile Object ID - ssm_file_id_t | 8-147 |
| 8.3.2.12. | Log File Object ID - ssm_logfile_t | 8-147 |
| 8.3.2.13. | Storage Server PV Object ID - ssm_ss_pv_t..... | 8-148 |
| 8.3.2.14. | Site List – SiteList_t..... | 8-148 |
| 8.3.2.15. | File Family Structure – FileFamilyStruct_t..... | 8-149 |
| 8.3.2.16. | File Family List – FileFamilyList_t..... | 8-149 |
| 8.3.2.17. | File Attribute Structure – ssm_fileattr_t..... | 8-149 |

| | | |
|-----------|--|------------|
| 8.3.2.18. | Logging Daemon Logfile Name Structure – ssm_logfile_t..... | 8-150 |
| 8.3.2.19. | HDM Fileset Identification Structure – ssm_hdm_fileset_id_t. | 8-150 |
| 8.3.2.20. | Name Server Fileset Information Structure – ssm_ns_fileset_t..... | 8-151 |
| 8.3.2.21. | Fileset Name Structure – ssm_fileset_name_t..... | 8-151 |
| 8.3.2.22. | DMAP Gateway Fileset Structure – ssm_dmg_fileset_t..... | 8-151 |
| 8.4. | Data Server Client Interfaces..... | 8-153 |
| 8.4.1. | client_Notify..... | 8-154 |
| 8.5. | Other Data Definitions (Data Server Clients)..... | 8-156 |
| 8.5.1. | Data Server Notification structure - NotifyUnion_t..... | 8-156 |
| 9. | Location Server Functions | 9-1 |
| 9.1. | Client Cache Programming Interface Functions..... | 9-1 |
| 9.1.1. | hpss_LocateBFSByCOSHints..... | 9-2 |
| 9.1.2. | hpss_LocateLocationServer..... | 9-4 |
| 9.1.3. | hpss_LocateRootNS..... | 9-5 |
| 9.1.4. | hpss_LocateServerByPath..... | 9-6 |
| 9.1.5. | hpss_LocateServerByUUID..... | 9-7 |
| 9.1.6. | hpss_LocationLibInit..... | 9-8 |
| 9.1.7. | hpss_LocationLibDeinit..... | 9-9 |
| 9.1.8. | hpss_LocationLibGetConfig..... | 9-10 |
| 9.1.9. | hpss_LocationLibSetConfig..... | 9-11 |
| 9.2. | Server Programming Interface Functions..... | 9-12 |
| 9.2.1. | ls_BFSByCOSHints..... | 9-13 |
| 9.2.2. | ls_GetServerMaps..... | 9-15 |
| 9.2.3. | ls_LocationServer..... | 9-17 |
| 9.2.4. | ls_RootNS..... | 9-18 |
| 9.2.5. | ls_ServerByPath..... | 9-19 |
| 9.2.6. | ls_ServerByUUID..... | 9-20 |
| 9.2.7. | ls_ServerGetAttrs..... | 9-21 |
| 9.2.8. | ls_ServerSetAttrs..... | 9-22 |
| 9.2.9. | ls_StatGetAttrs..... | 9-23 |
| 9.2.10. | ls_StatSetAttrs..... | 9-24 |
| 9.3. | Data Definitions..... | 9-25 |
| 9.3.1. | Location Map Structure – ls_map_t..... | 9-25 |
| 9.3.2. | Location Map Array – ls_map_array_t..... | 9-26 |
| 9.3.3. | COS/BFS Selection Structure – ls_cos_bfs_t..... | 9-26 |
| 9.3.4. | COS/BFS Array Structure – ls_cos_bfs_array_t..... | 9-27 |
| 9.3.5. | Location Server Statistics Structure – ls_server_stats_t..... | 9-27 |
| 9.3.6. | Location Policy Metadata Structure – ls_policy_md_t..... | 9-30 |
| 9.3.7. | Remote HPSS Site Metadata Structure – hpss_site_md_t..... | 9-31 |
| | Appendix A - Acronyms..... | A-1 |
| | Appendix B - References..... | B-1 |

Preface

This High Performance Storage System (HPSS) Programmer's Reference Guide, Volume 2 - Release 4.1.1, documents core server function calls which are provided by HPSS. It is designed for systems programmers.

HPSS provides an open interface with application programming interfaces to each HPSS server. Volume 2 documents these function calls interfaces to the core HPSS servers. While it is envisioned that most users will access HPSS through the client API, standard FTP, parallel FTP, NFS, DFS, MPI-IO or the Parallel I/O File System Import / Export interfaces, well defined programming interfaces are also defined to each HPSS server. It should be noted that programming to the individual server level is a more complex programming model which requires a greater understanding of the HPSS servers.

It is beyond the scope of this document to provide detailed information on programming at the individual server level. The API specifications and related data structures are documented for the core HPSS servers. However, it should be realized that programming at the inter-server level will require more of a working knowledge of HPSS internals than the Client APIs documented in Volume 1. In addition, internal infrastructure APIs (e.g. logging, metadata manager, DCE services, communications), and APIs for those servers which are unlikely candidates for application programming (e.g. Storage System Manager, Migration Purge Server) are not included in this document.

The objective of this document is to meet the following general goals:

- Define any known limitations of the APIs
- Define the HPSS server application programming interfaces (APIs).
- Define the data definitions referenced by the APIs

Refer to the HPSS Programmer's Reference Volume 1 for programming interfaces provided to the end user. Refer to the HPSS User's Guide for command interfaces provided to end users.

Refer to the HPSS User's Guide for a description of the following command line interfaces: standard FTP, parallel FTP, NFS, DFS, IBM SP Parallel I/O File System Import / Export, and user utilities.

Refer to the HPSS Error Messages Manual for a list of all HPSS error and advisory messages which are output by the HPSS software. For each message, the following information is provided: message identifier and text, source file name(s) which generated the message, problem description, system action, and administrator action.

Refer to the HPSS Administration Guide for a description of the interfaces provided to HPSS administrators.

This HPSS Programmer's Reference Guide, Volume 2 is structured as follows:

Chapter 1: Overview

This chapter provides an overview of each core server programming interface, constraints, and required libraries.

Chapter 2: Name Server Functions

This chapter defines the Name Server API specifications and associated data definitions..

| | |
|--|---|
| Chapter 3: Bitfile Server Functions | This chapter defines the Name Server API specifications and associated data definitions.. |
| Chapter 4: Storage Server Functions | This chapter defines the Name Server API specifications and associated data definitions.. |
| Chapter 5: Mover Functions | This chapter defines the Mover API specifications and associated data definitions.. |
| Chapter 6: Physical Volume Library Functions | This chapter defines the Physical Volume Library API specifications and associated data definitions.. |
| Chapter 7: Physical Volume Repository Functions | This chapter defines the Physical Volume Repository specifications and associated data definitions.. |
| Chapter 8: Storage System Manager Functions | This chapter defines the System Manager API specifications and associated data definitions. |
| Chapter 9: Location Server Functions | This chapter defines the Location Server API specifications and associated data definitions. |
| Appendix A: Acronyms | This appendix provides a list of acronyms used document. |
| Appendix B: References | This appendix lists documents cited in the text as other reference material. |

Typographic and Keying Conventions

This document uses the following typographic conventions:

| | |
|---------------|---|
| Bold | Bold words or characters represent system elements that you must use literally, such as functions, commands or keywords. |
| <i>Italic</i> | <i>Italic</i> words or characters represent variable values to be supplied. |
| [] | Brackets enclose optional items in syntax and format descriptions. |
| { } | Braces enclose a list of items to select in syntax and format descriptions. |

1. Overview

The High Performance Storage System (HPSS) provides scalable parallel storage systems for highly parallel computers as well as traditional supercomputers and workstation clusters. Concentrating on meeting the high end of storage system and data management requirements, HPSS is scalable and is designed for large storage capacities, and to use network-connected storage devices to transfer data at rates up to multiple gigabytes per second. Listed below is a description of the core HPSS servers.

1.1. Name Server

1.1.1. Purpose

The purpose of the Name Server is to map a name to an HPSS object. Names are generally human readable ASCII strings of 255 characters or less. Objects are files, directories, junctions, filesets, or links (symbolic links and hard links). In addition to mapping names to objects, the Name Server provides access verification to objects. The implementation defined in this design document provides a POSIX view of the name space which is a hierarchical structure consisting of directories, files, junctions and links.

1.1.2. Components

The Name Server uses a layered approach to inter-routine relationships. The software is layered as defined below:

- Remote Interface Routines (RIR)
- Local Interface Routines (LIR)
- Database Interface Routines (DIR)
- System Interface Routines (SIR)

The RIR layer handles transaction processing, security functions and translates remotely invoked functions to the appropriate local interface routine. This layer is also responsible for parsing path names and implementing the "." and ".." directories. The LIR handle the requested function and make use of the DIR layer to retrieve and store directory object metadata. The DIR layer makes use of Transarc's Encina Structured File System.

1.1.3. Constraints

The following constraints are being imposed upon HPSS as a result of this subsystem design:

- Hard links are only supported for files.

1.1.4. Libraries

Applications calling the Name Server function calls must link with the following libraries:

libmetadata.a
libhpsscs.a
libhpsscomm.a

libhpslog.a
libgss.a
libhsec.a
libhandles.a
libEncina.a
libEncClient.a
libEncSfs.a
libdce.a
libpthreads.a

1.2. Bitfile Server

1.2.1. Purpose

The Bitfile Server provides the abstraction of logical bitfiles to its clients. A logical bitfile is a bit string that is unconstrained in size and structure. A bitfile is identified by a Bitfile Server generated name called a bitfile ID. Mapping of a human readable name to the bitfile ID must be provided by a Name Server external to the Bitfile Server. Clients may reference portions of a bitfile by specifying the bitfile ID and a starting address and length. The *writes* and *reads* to a bitfile are random and the writes may leave "holes" where no data has been written. The Bitfile Server supports the parallel read and write of data to bitfiles. In conjunction with Storage Servers, the Bitfile Server maps logical portions of bitfiles onto physical storage devices.

The Bitfile Server provides commands to allow the migration, purging, and staging of data in a storage hierarchy.

1.2.2. Bitfile Server Components

The Bitfile Server consists of these major parts:

- Initialization
- Client APIs
- Storage System Management APIs.

Initialization starts up the Bitfile Server, makes connections needed to other servers, and sets up internal tables.

Client APIs are essentially the user interface to the Bitfile Server. They allow bitfiles to be created, stored, read, and allow bitfile attributes to be read and set.

Various APIs are used by both clients and SSM.

1.2.3. Constraints

The following constraints are being imposed upon HPSS as a result of this subsystem design:

- All transfer requests are for single bitfiles only. The multiple bitfiles allowed by the IOD will not be supported.

- Files that are highly fragmented will cause system performance to be degraded.
- The bitfile must be open to do reads, writes, migrates, stages, purges, and various options of get and set attributes.
- A reverse map field of all binary zeros is considered to be a null reverse map.

1.2.4. Libraries

Applications calling the Bitfile Server function calls must link with the following libraries:

libmetadata.a
libhpsscs.a
libhpsscomm.a
libhpsslog.a
libhpssgss.a
libhsec.a
libhandles.a
libtraniod.a
libEncina.a
libEncClient.a
libEncSfs.a
libdce.a
libpthreads.a

1.3. Storage Server

1.3.1. Purpose

The Storage Server provides a hierarchy of storage objects: storage segments, virtual volumes and physical volumes. The server translates references to storage segments into references to virtual volumes, and finally into physical volume references. It also schedules the mounting and dismounting of removable media. Clients of the Storage Server will be the HPSS Bitfile Server at the segment interface and the HPSS Storage System Manager at the virtual and physical volume interface.

1.3.2. Components

The Storage Server consists of these major parts:

- storage segment service
- virtual volume service
- physical volume service

The storage segment service is the conventional method for obtaining and accessing HPSS storage resources. The server maps an abstract storage space, the storage segment, onto a virtual volume, resolving segment addresses as required. The client is presented with a storage address space, with addresses from 0 to N-1, where N is the byte length of the segment. Segments can be opened, created, read, written, closed and deleted. Characteristics and information about segments can be retrieved and changed.

The virtual volume service is the method provided by the Storage Server to group physical storage volumes. The server maps the virtual volume address space onto the component physical volumes in a fashion appropriate to the grouping. The client is presented with a virtual volume that can be addressed from 0 to N-1, where N is the byte length of the virtual volume. Virtual volumes can be mounted, created, read, written, unmounted and deleted. Characteristics of the volume can be retrieved and in some cases, changed.

The physical volume service is the method provided by the Storage Server to access the physical storage volumes in HPSS. Physical volumes can be mounted, created, read, written, unmounted and deleted. Characteristics of the volume can be retrieved and in some cases, changed.

All three layers of the Storage Server can be accessed by appropriately privileged clients.

1.3.3. Constraints

The following constraints are being imposed upon HPSS as a result of this subsystem design:

- A storage segment cannot span virtual volumes.
- A physical volume cannot span multiple virtual volumes.
- Intermediate IORs for I/O requests will not be generated or provided by the Storage Server. I/O functions (read and write) are synchronous (e.g. They do not reply until the I/O is complete; however, it is possible for the client to issue parallel I/O requests to the server).

1.3.4. Libraries

Applications calling the Storage Server function calls must link with the following libraries:

- libmetadata.a**
- libhpsscs.a**
- libhpsscomm.a**
- libhpsslog.a**
- libhpssgss.a**
- libhsec.a**
- libhandles.a**
- libtraniod.a**
- libgssmvr.a**
- libpdata.a**
- libpvl.a**
- libEncina.a**
- libEncClient.a**
- libEncSfs.a**
- libdce.a**
- libpthreads.a**

1.4. Mover

1.4.1. Purpose

The purpose of the Mover is to transfer data from a source device to a sink device. A device can be a standard I/O device with geometry (e.g., tape, disk, optical disk), or a device without geometry (e.g., network, memory). The Mover will retry requests and attempt to optimize requests, but will not take any action that is outside the scope of what is requested by the Mover's clients.

Additional support is provided for:

Disk devices.

Third party IPI-3 data transfers.

Sending intermediate responses with listen port addressing information.

Using a Mover to Mover data transfer control protocol.

1.4.2. Components

The Mover consists of these major parts:

- Mover Parent Task
- Mover Listen Task / Request Processing Task
- Data Movement
- Device Control
- System Management

The Mover Parent Task performs some of the Mover initialization functions, and spawns processes to handle the Mover's DCE communications as well as the Mover's functional interface (which does not use DCE pthreads).

The Mover Listen Task listens on a well known TCP port for incoming connections to the Mover, spawns request processing tasks (forks processes in Releases 1 and 2), and monitors for completion of those tasks. The Request Processing Task performs initialization and return functions common to all Mover requests.

Data Movement supports client requests to transfer data to or from HPSS, and includes the **mvr_Read** and **mvr_Write** interfaces. The ability to abort an outstanding data movement request is provided via the **mvr_Abort** interface.

Device Control supports querying the current device read/write position (for use in a later search operation), changing the current device read/write position and performing device specific operations, and includes the **mvr_DeviceGetAttrs_IOD**, **mvr_DeviceSetAttrs_IOD** and **mvr_DevSpec** interfaces.

System Management supports querying and altering device characteristics and overall Mover state, and includes the **mvr_MVRGetAttrs**, **mvr_MVRSetAttrs**, **mvr_DeviceGetAttrs**, **mvr_DeviceSetAttrs**, **mvr_ServerGetAttrs** and **mvr_ServerSetAttrs** interfaces. Also supported is adding new devices and removing existing devices via the **mvr_CreateDevice** and **mvr_DeleteDevice** interfaces.

1.4.3. Constraints

The following constraints are being imposed upon HPSS as a result of this subsystem design:

- Due to conflicts between DCE and asynchronous I/O (in particular using DCE results in the possibility of lost signals), the Mover data transfer and device positioning code will not use either DCE RPC or DCE pthreads. Instead, the Mover will use the DCE marshalling routines and transfer requests and replies over TCP streams.
- A process that utilizes DCE RPC and Pthreads will be spawned at Mover initialization to handle non I/O requests (e.g., Mover state requests). Note that this requires that all machines running an HPSS Mover also run DCE and Encina.

1.4.4. Libraries

Applications calling the Mover function calls must link with the following libraries:

libmetadata.a
libhpsscs.a
libhpsscomm.a
libhpsslog.a
libhpssgss.a
libhsec.a
libhandles.a
libEncina.a
libEncClient.a
libEncSfs.a
libdce.a
libpthreads.a

1.5. Physical Volume Library

1.5.1. Purpose

The PVL manages all HPSS physical volumes. Clients can ask the PVL to mount and dismount sets of physical volumes. Clients can also query the status and characteristics of physical volumes. The PVL maintains a mapping of physical volume to cartridge and a mapping of cartridge to PVR. The PVL also controls all allocation of drives. When the PVL accepts client requests for volume mounts, the PVL allocates resources to satisfy the request. When all resources are available, the PVL issues commands to the PVR(s) to mount cartridges in drives. The client is notified when the mount has completed.

1.5.2. Components

The PVL consists of these major parts:

- Volume mount service
- Storage system management service

The volume mount service is provided to clients like a Storage Server. Multiple volumes may be specified as part of a single request. All of the volumes will be mounted before the request is satisfied. All volume mount requests from all clients are handled by the PVL. This allows the PVL to prevent multiple clients from deadlocking when trying to mount intersecting sets of volumes. The standard mount interface is asynchronous. A notification is provided to the client when the entire set of volumes has been mounted. A synchronous mount interface is also provided. The synchronous interface can only be used to mount a

single volume, not sets of volumes. The synchronous interface might be used by a non-HPSS process to mount cartridges which are in a tape library, but not part of the HPSS system.

The storage system management service is provided to allow a management client control over HPSS tape repositories. Interfaces are provided to import, export, and move volumes. When volumes are imported into HPSS, the PVL is responsible for writing a label to the volume. This label can be used to confirm the identity of the volume every time it is mounted. Management interfaces are also provided to query and set the status of all hardware managed by the PVL (volumes, drives, and repositories).

1.5.3. Constraints

The following constraints are being imposed upon HPSS as a result of this subsystem design:

- No attempt is made to optimize volume mounts. They are satisfied on a first come, first served basis. If a volume is mounted before it is requested by the PVL it may be used out of the normal order unless the PVL determines that such use might result in a deadlock.
- Volume names are derived from the cartridge name and the side of the cartridge. Cartridge names must be unique across an entire HPSS installation.

1.5.4. Libraries

Applications calling the Physical Volume Library function calls must link with the following libraries:

libpvl.a
libmetadata.a
libhpsscs.a
libhpsscomm.a
libhpsslog.a
libhpssgssmvr.a
libhsec.a
libhandles.a
libEncina.a
libEncClient.a
libEncSfs.a
libdce.a
libpthreads.a

1.6. Physical Volume Repository

1.6.1. Purpose

The PVR manages all HPSS cartridges. Clients can ask the PVR to mount and dismount cartridges. Every cartridge in HPSS must be managed by exactly one PVR. Clients can also query the status and characteristics of cartridges.

1.6.2. Components

The PVR consists of these major parts:

- Generic PVR service
- Ampex robot service
- STK robot service
- 3494 / 3495 robot service
- Operator mounted device service

The generic PVR service provides a common set of APIs to the client regardless of the type of mount device being managed. Functions to mount, dismount, inject, and eject cartridges are provided. Additional functions to query and set cartridge metadata are provided. The mount function is asynchronous. The PVR calls a well-known API in the client when the mount has completed. For certain devices, like operator mounted repositories, the PVR will not know when the mount has completed. In this case, it is up to the client to determine when the mount has completed. The client may poll the devices or use some other method. When the client determines a mount has completed, the client should notify the PVR using one of the PVR's APIs. All other PVR functions are synchronous. The generic PVR maintains metadata for each cartridge managed by the PVR.

The Ampex robot service manages the Ampex DST 800 robotic device. This device mounts, dismounts, and manages D2 cartridges for a set of Ampex D2 drives. The Ampex robot service maintains additional metadata about each cartridge it manages.

The STK robot service manages the STK Silo robotic device. This device mounts, dismounts, and manages 3480 / 3490 cartridges for a set of 3480 / 3490 drives. The STK robot service maintains additional metadata about each cartridge it manages.

The 3494 / 3495 robot service manages the two IBM tape robots. These robots manage 3480 form factor cartridges. The cartridges may be for 3480, 3490, or 3590 type drives. The robots, while physically very different, are managed through virtually identical interfaces.

The operator mounted device service manages a set of cartridges that are not under the control of a robotic device. These cartridges are mounted to a set of drives by operators. The Storage System Manager is used to inform the operators when mount operations are required.

1.6.3. Constraints

The following constraints are being imposed upon HPSS as a result of this subsystem design:

- It is expected that the PVR's clients will be able to determine when cartridges are mounted. This should be done with polling or some asynchronous notification. The client should also be able to accept asynchronous notifications from the PVR for those times when the PVR is able to determine that a cartridge is mounted.

1.6.4. Libraries

Applications calling the Physical Volume Repository function calls must link with the following libraries:

libpvr.a
libmetadata.a
libhpsscs.a
libhpsscomm.a
libhpsslog.a

libhpssgss.a
libhsec.a
libhandles.a
libEncina.a
libEncClient.a
libEncSfs.a
libdce.a
libpthreads.a

1.7. System Manager

1.7.1. Purpose

The SSM System Manager is the contact point between clients, such as the SSM Data Server (which is the graphical interface to the human operator or system administrator), and the other HPSS subsystems. Interfaces are provided to support external clients, in addition to the Data Server. The term Data Server will be used to refer to the HPSS provided Data Server or other external clients of the System Manager.

All Data Server requests to other HPSS servers and all Data Server Encina accesses are made on the client's behalf by the System Manager. Operations provided by the System Manager to the Data Server include configuration of Encina files, starting and shutting down servers, importing and exporting media, control of devices and jobs, viewing and updating managed objects, and delogging.

All alarms, events, status messages, and notifications issued to SSM by other subsystems are received by the System Manager and forwarded to the Data Server as appropriate, using the Data Server **client_Notify** API.

The System Manager also uses the client_Notify API to notify the Data Server of changes in SSM data or state, such as a change in the SSM Server List or a warning that the System Manager is shutting down.

1.7.2. Components

The System Manager consists of these major parts:

- Initialization
- System Manager Client Support
- Configuration
- Administrative Operations
- Managed Object Attribute Operations
- Device Management
- Job Management
- Delogging
- Storage and Media Operations

- Accounting
- Alarm, Event, and Status Message Processing

Initialization starts up the Bitfile Server, makes connections needed to other servers, and sets up internal tables.

Initialization

At startup, the System Manager reads the HPSS Server Configuration File and builds a copy of it in `SSM_SM_server_config`. From this copy, it builds the Server List `SSM_SM_servers` and initializes the Server Network Connection Table `SSM_SM_server_net`. The Server List includes information needed by the System Manager and the Data Server such as the descriptive name, `uuid`, server type, and execution status of each server. The Server Network Connection Table includes information needed by the System Manager to connect to each server, including interface specifications, binding handles, and connection handles. At this point in startup, only the interface specification is defined; binding handles are deferred.

The System Manager next reads the necessary configuration files to build the other lists it shares with its clients. It reads the Mover Device Configuration File and the PVL Drive Configuration File and builds from the combined information from both files the Drive List, `SSM_SM_drives`, which contains information needed by the System Manager and the Data Server such as the device and drive name and the associated PVL, PVR, and Mover for each drive.

It reads the Class of Service Configuration File, the Storage Class Configuration File, the Hierarchy Configuration File, the Migration Policy Configuration File, and the Purge Policy Configuration File and constructs the Class of Service List, The Storage Class List, the Hierarchy List, the Migration Policy List, and the Purge Policy List, which are needed by the Data Server for building selection lists and for managing the storage class window.

Next the System Manager spawns the Server List monitor thread. In most cases, the function which changes a list will enqueue a notification to the Data Server about the change, so every single change to the list will result in a notification to the Data Server. In the case of the Server List, which changes very frequently, a monitor thread is created which checks the list periodically and forwards it to the Data Server if it has changed, so several changes might be made to the list before a copy is forwarded to the Data Server.

Next the System Manager spawns a separate thread for each server to monitor that server's execution and connection status.

Finally, the System Manager registers its interfaces and enters a `trdce_ServerListen`.

System Manager Client Support

Data Servers make contact with the System Manager with `ssm_CheckIn`, using the input `ClientID` `SSM_NEW_CLIENT` to indicate an initial check-in. The System Manager returns a unique output `ClientID` and then sends the new client a copy of each of the shared lists in separate notifications.

The client may check-in again with the System Manager at any time using the `ClientID` it was assigned at its initial check-in. This should always be done when the client has temporarily lost and then regained network connectivity to the System Manager, first in order to get a current copy of all the shared lists, and second to make certain the System Manager still recognizes the client. If the System Manager crashed and restarted, for instance, it will not know about the client and will return a failure on the subsequent check-in; the client should then repeat its initial check-in using `SSM_NEW_CLIENT` as its `InClientID`.

Clients discontinue contact with the System Manager by calling `ssm_CheckOut`. If the System Manager loses contact with a Data Server for more than `SSM_SM_CLIENT_MAX_FAILTIME` seconds, it will

automatically check him out.

Whenever the System Manager receives notifications of alarms, events, status messages, changes in managed object attributes, tape mount requests or tape check-in requests, it forwards these to the appropriate clients using the **client_Notify** API. Whenever one of the shared lists changes, the System Manager informs each Data Server by calling **client_Notify** with an SSM_LIST_N type notification containing the appropriate updated list.

Notifications are queued to avoid flooding the Data Server, which results in losing contact with it. There are five notification queues:

| | |
|------------------------------|---|
| SSM_SM_notify_q_data | managed object attribute changes. |
| SSM_SM_notify_q_list | list changes and informational notifications. |
| SSM_SM_notify_q_log | alarms, events, status messages. |
| SM_SM_notify_q_tape | tape mount notifications. |
| SSM_SM_notify_q_tape_checkin | tape check-in notifications. |

Since there is only one kind of informational notification and it is only used as the System Manager is shutting down, it was combined with the list queue.

Incoming notifications are throttled in order to keep the System Manager memory usage from growing too fast. When the queue reaches a certain limit, the notification function waits till it shrinks before adding the new item to the queue.

Configuration

HPSS servers store permanent data about server, device, media, and policy configuration in Encina configuration files. With the following APIs a Data Server may request the System Manager to read and update HPSS configuration files:

| | |
|-----------------------------|--|
| ssm_ConfigAdd | Adds one entry to a file. |
| ssm_ConfigDelete | Deletes one entry from a file. |
| ssm_ConfigGetDefault | Returns a default configuration file entry of the type requested. The Data Server calls ssm_ConfigGetDefault to obtain default data as a starting point whenever the user asks to add a new entry to a file. |
| ssm_ConfigRead | Reads the specified entry from Encina and returns it. |
| ssm_ConfigUpdate | Modifies one entry in a file. |

SSM does not have permission to write to all configuration files.

Some subsystems require notification whenever their configuration files change in the form of an ST_REINIT to their server managed object Administrative State. Some subsystems require that SSM not change their configuration at all while they are executing. The configuration APIs take the appropriate action for each server.

Administrative Operations

Administrative operations are provided to the Data Server by the **ssm_Adm** function and include:

Starting one or all servers

- Reinitializing one or all servers
- Shutting down one or all servers
- Forcing a halt of one or all servers
- Setting a server's state to REPAIRED
- Forcing connection to a server
- Shutting down HPSS

To an extent, many of these are functions of setting the Administrative State attribute on the server managed object. Changes to managed objects are normally handled by **ssm_AttrSet**. However, the System Manager requires that changes to a server's Administrative State be made through the **ssm_Adm** function in order to make it easier to do any special processing required for the change. For example, halting a server involves first setting his Administrative State to ST_HALT, but most servers never return from such a request, as they shut down immediately. The System Manager must then ask the startup demon whether the server is still running, and ask him to kill the server if he is .

The Repair function is provided to enable the human operator to inform the server that some error condition previously reported by the server has been corrected. The subsequent action taken by the server is up to that server, but in general the server is expected to reexamine the area of error and clear its associated error flags accordingly.

The "Connect to server" function is provided to allow the operator to ask the System Manager to attempt connection immediately to the specified server. The System Manager will automatically check connections at a specified interval.

Managed Object Attribute Operations

While they are executing, HPSS servers store current and volatile data about server, device, and media configuration in data structures called managed objects. In some cases, the same data structure is used to define both the managed object and the corresponding Encina configuration file entry; in other cases the attributes defined for the managed object overlap those defined for the configuration file; in still other cases there is no corresponding managed object for a configuration file. In general, the most current information about an entity is to be found by asking the server about its managed object rather than by reading the configuration file.

Managed object attributes may be viewed by a Data Server with **ssm_AttrGet** and modified by **ssm_AttrSet**. The **ssm_AttrReg** function allows a Data Server to register to receive notifications of changes in specified attributes of a managed object.

In practice, a GUI Data Server uses **ssm_AttrReg** whenever a user opens a managed object window, so that he can keep the window refreshed with the latest information from the server. A Data Server also uses **ssm_AttrReg** to register for the *OpState* on servers, drives, and the generic volumes, so he can monitor the status of the system. A GUI Data Server may use **ssm_AttrGet** to poll certain servers for statistics, and **ssm_AttrSet** when users modify writeable fields on managed object windows.

The System Manager maintains a table of the attributes for which each Data Server is registered, SSM_SM_registered_mo. When it receives a data change notification from a server, it searches this table and notifies the clients who are registered to receive that notification using the **client_Notify** API.

The APIs with which servers notify the System Manager of managed object attribute changes are:

| API: | Server: | Managed Object: |
|--------------------------|------------------|-----------------|
| ssm_BitfileNotify | BFS | bitfile |
| ssm_LogFileNotify | Log Daemon | logfile |
| ssm_SFSNotify | Metadata Monitor | SFS |

| | | |
|-----------------------------|----------------|-----------------|
| ssm_MPSNotify | MPS | mps |
| ssm_MPS_SClassNotify | MPS | storage class |
| ssm_DeviceNotify | Mover | device |
| ssm_MVRNotify | Mover | mover |
| ssm_NFS2_StatsNotify | NFS Daemon | nfs statistics |
| ssm_NSNotify | Name Server | name server |
| ssm_DriveNotify | PVL | drive |
| ssm_PVLNotify | PVL | pvl |
| ssm_QueueNotify | PVL | queue |
| ssm_RequestNotify | PVL | request |
| ssm_VolNotify | PVL | volume |
| ssm_CartNotify | PVR | cartridge |
| ssm_PVRNotify | PVR | pvr |
| ssm_ServerNotify | All | server |
| ssm_MapNotify | Storage Server | storage map |
| ssm_PVNotify | Storage Server | physical volume |
| ssm_SSNotify | Storage Server | storage segment |
| ssm_SsrvNotify | Storage Server | storage server |
| ssm_VVNotify | Storage Server | virtual volume |

Device Management

Device Management operations include viewing device information, varying drives online and offline, forcing drive dismounts, and relaying mount request information.

Viewing device information is accomplished by calling the **ssm_AttrGet** or **ssm_AttrReg** function for both the Mover device managed object and the PVL drive managed object.

Varying drives online and offline is accomplished by setting the Administrative State of the PVL drive managed object to ST_UNLOCKED or ST_LOCKED, respectively, and so is accomplished by calling **ssm_AttrSet**.

The **ssm_DriveDismount** function enables the operator to force a dismount of a drive in the event the PVR does not automatically perform the dismount.

The PVR sends the System Manager notifications of mount requests and mount completions so that mount requests for human-operated PVRs can be displayed and so that mount requests for robot-operated PVRs which get stuck can be noticed. The System Manager receives these notifications with **ssm_MountNotify**, and forwards them to all Data Servers using the client_Notify API. The PVR also sends the System Manager tape check-in notifications to display a list of cartridges for the operator to insert into the I/O port. The System Manager receives these notifications with **ssm_TapeCheckInNotify**, and forwards them to all Data Servers using the client_Notify API.

Job Management

Job management operations include displaying the job queue and canceling jobs.

Viewing the job queue is accomplished by calling **ssm_AttrGet** or **ssm_AttrReg** for the PVL queue managed object.

Canceling jobs is performed by calling **ssm_JobCancel**.

Delogging

The **ssm_Delogg** function retrieves selected records of the HPSS alarm and event log and places the output in a UNIX file. A GUI Data Server might open a window to display the file and allow the operator to browse it. For this to work, the UNIX file must be accessible by both the System Manager, which executes the delog program, and by Sammi, which opens the window to view the file.

Storage and Media Operations

The storage and media operations allow the operator to view attributes of cartridges and the storage data structures built upon them and to define cartridges to the PVL and the Storage Server.

Viewing attributes of a cartridge is a function of getting the PVR cartridge managed object attributes and is performed by calling **ssm_AttrGet** or **ssm_AttrReg**.

Defining cartridges to the PVL is performed with **ssm_CartImport**, and removing the cartridges from the PVL is performed with **ssm_CartExport**. Cartridges may be moved from one PVR to another with **ssm_CartMove**.

Defining cartridges to the Storage Server means defining the Storage Server resource management data structures for the cartridges, which include physical volumes, virtual volumes, and storage segment maps. It is accomplished by calling **ssm_ResourceCreate**. The structure definitions are removed from the Storage Server with the **ssm_ResourceDelete** function. Volumes may be repacked with **ssm_ResourceRepack** and reclaimed with **ssm_ResourceReclaim**. Repack and reclaim are implemented very minimally from SSM in the current release; not all the options supplied by the command-line programs are available from SSM.

Functions not implemented in the current release include deleting inactive storage maps, listing all cartridges, labeling cartridges, auditing the PVR, dismounting physical volumes, and dismounting cartridges.

Accounting

The **ssm_AcctRun** API starts an execution of the accounting program. The **ssm_AcctChange** API changes the account id on a specified bitfile.

Alarm, Event, and Status Message Processing

The logger sends selected alarms, events, and status messages to the System Manager based on the settings in the HPSS Log Policy File. The System Manager forwards all received alarms, events, and status messages to all Data Servers using the **client_Notify** API.

The API with which the logger notifies the System Manager for all three types of message is **ssm_LogMsgNotify**.

1.7.3. Constraints

The following constraints are being imposed upon HPSS as a result of this subsystem design:

- Data Server Clients, those programming to the APIs provided in the **ssm_client_if** interface, must run under a principal which has control permission on the System Managers's Security Object. Other clients, those programming only to the notification APIs, do not require control permission.

Data Server Clients are expected to provide the System Manager a **client_Notify** API to receive

asynchronous notifications. Clients who do not provide this api will be automatically checked out by the System Manager.

1.7.4. Libraries

Applications calling the System Manager function calls must link with the following libraries:

libmetadata.a
libhpsscs.a
libhpsscomm.a
libhpsslog.a
libhpssgss.a
libhsec.a
libhandles.a
libEncina.a
libEncClient.a
libEncSfs.a
libdce.a
libpthreads.a

1.8. Location Server

1.8.1. Purpose

The purpose of the Location Server (LS) is to provide a service which allows various HPSS servers to locate other HPSS servers both in the local site and at remote sites. The Location Server also provides Class of Service (COS) selection to the HPSS Client API by maintaining local COS statistics obtained from local Bitfile (BFS) servers.

1.8.2. Components

The Location Server consists of two major parts:

- Client cache library
- Server interface

The Client Cache Library (CCL) provides access for a client (such as the Client API) to the Location Server's Client Interface through a client side cache. This allows a client to access Location Server information while reducing network traffic. An additional benefit of the CCL is that it performs automatic retries of client requests and randomly rebinds to replicated Location Servers as needed. Functions are included which map between Server UUIDs and Locations, locate BFSs by using COS hints, locate the local root Name Server and locate remote Location Servers by site. The CCL functions provided are contained in a library that is totally separate from the Location Server code.

The Server Interface provides the same functions of the CCL which allow clients to locate server information. These are normally accessed through the CCL. In addition, there are functions which allow remote Location Servers to exchange information as well as administrative functions allowing control of the Location Server itself.

1.8.3. Constraints

The following constraints are being imposed upon HPSS as a result of this subsystem design:

- When metadata that is used to perform Class of Service (COS) selection is modified, such as Class of Service, Hierarchy and Storage Class information, the Location Server must be recycled along with the Bitfile Server in order for the changes to take effect. This allows the LS and BFS to remain synchronized with respect to this information in order to perform COS selection properly during file creation.

1.8.4. Libraries

Applications calling the Location Client Cache Library function calls must link with the following libraries:

libgss_nolog.a
libhandles_nolog.a
libhpsscomm_nolog.a
libhpsscs.a
libhsec_nolog.a
libhpsslog.a
libhpssls.a

1.8.5. Referenced Data Types

The majority of the key data types from each of the core HPSS servers are described in the server chapters which follow. Some infrastructure and support data types are referenced, but not explicitly described. Those data types whose format is not described in this document may be found in the following HPSS header files:

| Data Type | File |
|---------------------------------|--|
| acct_config_t | acct_config.h |
| acct_rec_t | acct_hpss.h |
| bfs_config_info_t | bfs_config.h |
| bfs_lock_cb_t | bfs_lock.h |
| bf_schedl_info_t | bfs_cache.h |
| bf_tape_segment_cached_t | bfs_cache.h |
| cos_t | hpss_cos.h |
| hpsssem_t | hpsssem.h |
| hpssoid_t | hpssoid.h |
| hpss_connect_handle_t | hpssIF.h |
| hpss_fileattr_t | Refer to <i>HPSS Programmer's Reference Guide, Volume 1.</i> |
| hpss_migr_policy_md_t | hpss_migr_policy.h |
| hpss_object_handle_t | hpssIF.h |
| hpss_purge_policy_md_t | hpss_purge_policy.h |
| hpss_sclass_md_t | hpss_sclass.h |
| hpss_server_attr_t | hpss_server_attr.h |
| hpss_server_config_t | mm_idl_types.h |
| IOD_t | Refer to <i>HPSS Programmer's Reference Guide, Volume 1.</i> |
| IOR_t | Refer to <i>HPSS Programmer's Reference Guide, Volume 1.</i> |
| log_rec_hdr_t | cs_Log.h |
| LogFileAttr_t | cs_LogFileAttr.h |

| | |
|--------------------------|---------------------|
| LogcConfig_t | cs_LogcConfig.h |
| LogdConfig_t | cs_LogdConfig.h |
| LogPolicy_t | cs_LogPolicy.h |
| mm_mon_config_t | mm_idl_types.h |
| mountd_config_t | nfs2/mnt1_config.h |
| mps_attrib_t | mps_attrib_t.h |
| mps_config_t | mps_config.h |
| mps_sclass_attrs_array_t | mps_interface_def.h |
| mps_sclass_attrs_t | mps_interface_def.h |
| nfs2_stats_t | nfs2/nfs2_IFdefs.h |
| rpc_master_handle_t | hpss_rpc_handles.h |
| security_t | security.h |
| sfs_attrs_t | mm_ssm_data.h |
| timestamp_sec_t | hpss_idl_types.h |
| trpc_master_handle_t | hpss_trpc_handles.h |

2. Name Server Functions

This chapter specifies the Name Server programming interface. Specifically, the following information is provided:

Application Programming Interfaces (APIs)

Data Definitions

2.1. API Functions

This section describes all APIs which are provided for use by another HPSS subsystem or by a client external to HPSS. The API interface specification includes the following information:

Name

Syntax

Description

Parameters

Return Values

Error Conditions

Related Information

Clients

Notes

2.1.1. ns_Delete

Purpose

Delete a name space object.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_Delete (
```

```
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t        *UserCred,     /* IN */
    unsigned32             RequestID,     /* IN */
    ns_ObjHandle_t         *Directory,     /* IN */
    uchar                  *PathName,     /* IN */
    unsigned32             DontBackUp,     /* IN */
    unsigned32             Options,        /* IN */
    hpssoid_t              *BitFileId,     /* OUT */
    ns_ObjHandle_t         *FilesetHandle, /* OUT */
    unsigned32             *LinkCount,     /* OUT */
    ns_RemainingPath_t     *RemainingPath); /* OUT */
```

Description

This procedure deletes the object identified by *PathName* from the specified directory. The path name can identify a directory, a file, a hard link, or a junction. If no *PathName* is supplied, the object identified by *Directory* will be deleted. To delete a symbolic link, first fetch the object handle to the symbolic link and then submit the *ns_Delete* request with this object handle as the *Directory* and *PathName* set to NULL.

Parameters

| | |
|-------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory containing the object to be deleted. |
| <i>PathName</i> | Pointer to the path name of the object that is to be deleted. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any “..” components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow “..” path components to back up past the starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>Options</i> | Used to control the behavior of the delete. See the Notes below for the Option values. |

| | |
|----------------------|---|
| <i>BitFileId</i> | Pointer to a BitfileId. If the object being deleted is a file, the BitFileId to the file is returned. If the object is not a file this parameter will contain zeros. The <i>BitFileId</i> of a deleted file is returned so that the Client API can optimize its delete algorithm. |
| <i>FilesetHandle</i> | If the object being deleted is a junction, the object handle of the directory that the junction is pointing to is returned. If the object being deleted is not a junction, this parameter will contain zeros. |
| <i>LinkCount</i> | Pointer to a count of the number of objects still linked to the bitfile after the delete has occurred. |
| <i>RemainingPath</i> | Pointer to the structure containing returned information that is necessary to resolve the remainder of <i>PathName</i> . |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_Delete** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-----------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path or write permission is denied on the directory from which the object is to be deleted. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EBACKOVER | A PathName containing ".." components would have backed over the PathName origin and the DontBackUp option was set to 'true'. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EINVAL | The path name identifies a "." or ".." directory. |
| HPSS_EIO | An internal error occurred while reading from or writing to the Name Server data base. |
| HPSS_EISDIR | A attempt was made to delete a directory, but <i>Options</i> has the DONT_DELETE_DIR bit turned on. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |

| | |
|-------------------|--|
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component in the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component of the path name does not exist or the path name argument points to an empty string. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTEMPTY | The specified directory has entries (in addition to "." and ".."). |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the Directory is incorrect. |

See also

ns_Insert, **ns_Mkdir**, **ns_MkLink**, **ns_MkSymLink**, **ns_MkJunction**, and **ns_MkSymLink**.

Clients

Client APIs, *insif*.

Notes

The behavior of the delete is controlled by the *Options*. *Options* can one of the following values:

| | |
|--------------------|--|
| NS_DELETE_ANYTHING | Whatever type of object the <i>PathName</i> resolves to will be deleted. |
| NS_ONLY_DELETE_DIR | If the <i>PathName</i> resolves to an object type other than a directory, an error will be returned. |
| NS_DONT_DELETE_DIR | If the <i>PathName</i> does resolve to an object of type directory, an error will be returned. |

To delete a Symbolic Link object, provide an object handle.

2.1.2. ns_DeleteACL

Purpose

Delete a list of entries from the ACL of the specified Name Server object.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_DeleteACL(
    trpc_handle_t          BindH,          /* IN */
    hpss_object_handle_t  *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCred,     /* IN */
    unsigned32            RequestID,     /* IN */
    ns_ObjHandle_t        *Directory,     /* IN */
    uchar                 *PathName,     /* IN */
    unsigned32            DontBackUp,     /* IN */
    ns_ACLConfArray_t     *ACLEntries,   /* IN */
    ns_RemainingPath_t    *RemainingPath); /* OUT */
```

Description

Delete the specified entries from the ACL associated with the object specified by *PathName* and *Directory*. If no *PathName* is supplied, the object identified by *Directory* will be used.

Parameters

| | |
|-----------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory containing the object whose ACL entries are to be deleted. |
| <i>PathName</i> | Pointer to the path name of the object whose ACL entries are to be deleted. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any ".." components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow ".." path components to back up past the starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>ACLEntries</i> | Pointer to the array of ACL entries that are to be deleted. |
| <i>RemainingPathP</i> | Pointer to the structure containing returned information that is necessary to resolve the remainder of <i>PathName</i> . |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_DeleteACL** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-------------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path, or the owner does not have write permission, or others do not have control permission. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn’t convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from or writing to the Name Server data base. |
| HPSS_EMDATA | An inconsistency was encountered in the name server’s metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component in the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component of the path name does not exist. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESRCH | The corresponding ACLEntry was not found. |
| HPSS_ESTALE | The generation number in the Directory is incorrect. |

See also

ns_GetACL, ns_SetACL, ns_UpdateACL.

Clients

Client APIs, *insif*.

Notes

None.

2.1.3. ns_DeleteFileset

Purpose

Delete a fileset object.

Syntax

```
#include <cns_interface.h>
```

```
signed32
```

```
ns_DeleteFileset (
```

```
    trpc_handle_t          BindH,          /* IN */
```

```
    hpss_connect_handle_t *ConnectH,    /* IN */
```

```
    hsec_UserCred_t       *UserCreds,    /* IN */
```

```
    unsigned32            RequestId,     /* IN */
```

```
    ns_ObjHandle_t        *FilesetHandle, /* IN */
```

```
    u_signed64            *FilesetId);   /* IN */
```

Description

This transactional procedure deletes the fileset identified by either the *FilesetHandle* or the *FilesetId*.

Parameters

| | |
|----------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the <i>hpss_ConnectHandle</i> that defines the connection context for this user. |
| <i>UserCreds</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestId</i> | The request identifier specified by the client. |
| <i>FilesetHandle</i> | Pointer to a name server handle which describes the fileset to be deleted. If the <i>FilesetHandle</i> is NULL the <i>FilesetId</i> will be used to identify the object to be deleted. See the Notes below for more information. |
| <i>FilesetId</i> | Pointer to a fileset identifier which identifies the fileset to be deleted. If the <i>FilesetId</i> is NULL, the <i>FilesetHandle</i> will be used. See the Notes below for more information. |

Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

Error Conditions

The **ns_DeleteFileset** procedure is unsuccessful and the name server data base remains unchanged if any of the following are true:

| | |
|-------------|---|
| HPSS_EACCES | Write permission is denied. Only the Root user has permission to delete filesets. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |

| | |
|-----------------|--|
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The name server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from or writing to the name server data base. |
| HPSS_ENOTEMPTY | The specified fileset has entries (in addition to "." and ".."). |
| HPSS_ENOTREADY | The name server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the Directory is incorrect. |

See also

ns_MkFileset .

Clients

Client APIs, insif

Notes

If both the *FilesetHandle* and *FilesetId* are non-NULL the Name Server will insure that they both point to the same object. If they do not, an error will be returned. If both are NULL an error will be returned.

When the object handle is supplied additional overhead is incurred by having to read the object metadata file in addition to the fileset metadata file.

If the fileset is not empty the fileset will not be deleted and an error will be returned.

2.1.4. ns_GetACL

Purpose

Get the ACL for the specified Name Server object.

Syntax

```
#include "cns_interface.h"
signed32
ns_GetACL (
    trpc_handle_t          BindH,                /* IN */
    hpss_object_handle_t *ConnectH,            /* IN */
    hsec_UserCred_t       *UserCred,           /* IN */
    unsigned32            RequestID,           /* IN */
    ns_ObjHandle_t        *Directory,          /* IN */
    uchar                 *PathName,           /* IN */
    unsigned32            DontBackUp,          /* IN */
    ns_ACLConfArray_t     **ACLEntries,        /* OUT */
    ns_RemainingPath_t    *RemainingPath);     /* OUT */
```

Description

Get and return the ACL for the object identified by *PathName* located in the specified directory. If no *PathName* is supplied, the object identified by *Directory* is used.

Parameters

| | |
|-------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory containing the object whose ACL is to be obtained. |
| <i>PathName</i> | Pointer to the path name of the object whose ACL is to be obtained. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any ".." components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow ".." path components to back up past the starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>ACLEntries</i> | Pointer to the array of ACL entries. |

RemainingPathP Pointer to the structure containing returned information that is necessary to resolve the remainder of *PathName*. Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_GetACL** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-------------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A PathName containing “..” components would have backed over the PathName origin and the DontBackUp option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn’t convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EMDATA | An inconsistency was encountered in the name server’s metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from or writing to the Name Server data base. |
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component in the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component of the path name does not exist. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | he Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_DeleteACL, ns_SetACL, ns_UpdateACL.

Clients

Client APIs, insif

Notes

None.

2.1.5. ns_GetAttrs

Purpose

Get and return the Name Server handle, access ticket, and metadata (attributes) associated with the specified object.

Syntax

```
#include "cns_interface.h"
#include <hpssoid.h>

signed32
ns_GetAttrs (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCred,     /* IN */
    unsigned32            RequestID,     /* IN */
    ns_ObjHandle_t        *Directory,    /* IN */
    uchar                 *PathName,     /* IN */
    unsigned32            DontBackUp,    /* IN */
    unsigned32            ChaseSymLink,  /* IN */
    ns_AttrBits_t         ObjAttrBits,   /* IN */
    ns_AttrBits_t         ParentAttrBits, /* IN */
    ns_ObjHandle_t        *ObjHandle,    /* OUT */
    gss_token_t           *AcsTicket,    /* OUT */
    ns_Attrs_t            *ObjAttrs,     /* OUT */
    ns_Attrs_t            *ParentAttrs,  /* OUT */
    uchar                 *ObjName,     /* OUT */
    ns_RemainingPath_t    *RemainingPath); /* OUT */
```

Description

This non-transactional procedure is used to get and return the object handle, access ticket, name, and the attributes specified by *ObjAttrBits* for the object in the directory identified by *Directory* and *PathName*. If the *ParentAttrBits* are non-zero attributes for the parent directory are returned in *ParentAttrs*. **ns_GetAttrs** performs the POSIX `stat()` function for objects. If no *PathName* is specified attributes are returned for the object identified by *Directory*.

Parameters

| | |
|-------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory containing the object whose attributes are to be obtained. |
| <i>PathName</i> | Pointer to the path name of the object whose attributes are to be obtained. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any “..” components in the <i>PathName</i> would result in backing up |

past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow “..” path components to back up past the starting directory. If an error is returned, the RemainingPath will contain the remainder of the PathName.

| | |
|-----------------------|--|
| <i>ChaseSymLink</i> | If the value of <i>ChaseLink</i> is 1, and the last path component is a symbolic link, then the symbolic link will be interpreted until the final object is reached and the attributes of that object will be returned. If the value is 0, the attributes of the symbolic link itself will be returned. |
| <i>ObjAttrBits</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be returned in the attribute structure, <i>ObjAttrs</i> , below. |
| <i>ParentAttrBits</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be returned in the attribute structure, <i>ParentAttrs</i> , below. |
| <i>ObjHandle</i> | Pointer to the handle that identifies this object within the Name Server database. |
| <i>AcsTicket</i> | Pointer to a ticket that contains the access rights (determined from the ACLs) to the bitfile for the user identified in <i>UserCred</i> . If this pointer is null on input, a ticket will not be returned. The return argument will be null if a connection to the BFS cannot be established or the PathName refers to an object other than a file. |
| <i>ObjAttrs</i> | Pointer to a structure containing metadata information about the object. |
| <i>ParentAttrs</i> | Pointer to a structure containing metadata information about the parent of the object. |
| <i>ObjName</i> | Pointer to a string which contains the object name. |
| <i>RemainingPath</i> | Pointer to the structure containing the information necessary to resolve the remainder of <i>PathName</i> . |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_GetAttrs** routine is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-------------|---|
| HPSS_EACCES | Search permission is denied on a component of the directory path. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |

| | |
|-------------------|---|
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn’t convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EBADF | The bitfile identifier is not to a valid file. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from or writing to the Name Server data base. |
| HPSS_EMDATA | An inconsistency was encountered in the name server’s metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |
| HPSS_ENAMETOOLONG | The length of the path name exceeds <code>HPSS_MAX_PATH_NAME</code> or the length of a component in the path name exceeds <code>HPSS_MAX_FILE_NAME</code> . |
| HPSS_ENOENT | A component of the path name does not exist. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_SetAttr, **ns_ReadDir** and **ns_ReadLink**.

Clients

Client API, `insif`

Notes

ns_GetAttrs is used for three purposes: to obtain the Name Server handle for the object, to obtain an access ticket, and to obtain attributes about the object. When used to obtain an object handle or an access ticket, the bit vectors should be 0.

When the path name resolves to a directory, junction, fileset, or symbolic link, the authorization ticket that is returned will be NULL. The bitfile server’s BitfileId is returned through the *ObjAttrs* structure. In addition other attributes about the bitfile, such as its size and time last accessed, are also returned. Although attributes about the bitfile are returned, these attributes cannot be set using **ns_SetAttrs**. Bitfile attributes must be set using **bfs_SetAttrs**.

The reason the Name Server returns bitfile attributes is to improve the performance of the `ftp ls` command when the `-l` option is used.

2.1.6. ns_GetFilesetAttrs

Purpose

Get and return the metadata (attributes) associated with the specified fileset.

Syntax

```
#include <cns_interface.h>
signed32
```

```
ns_GetFilesetAttrs (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCreds,    /* IN */
    unsigned32            RequestId,     /* IN */
    ns_ObjHandle_t        *FilesetHandle, /* IN */
    u_signed64            *FilesetId,    /* IN */
    ns_FilesetAttrBits_t  AttrBits,     /* IN */
    ns_FilesetAttrs_t     *Attrs);      /* OUT */
```

Description

This non-transactional procedure is used to get and return the name server fileset attributes specified by *AttrBits* for the fileset specified by *FilesetHandle* or *FilesetId*.

Parameters

| | |
|----------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the <i>hpss_ConnectHandle</i> that defines the connection context for this user. |
| <i>UserCreds</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestId</i> | The request identifier specified by the client. |
| <i>FilesetHandle</i> | Pointer to an object handle which uniquely identifies the fileset whose attributes are to be returned. If the <i>FilesetHandle</i> is NULL the <i>FilesetId</i> will be used. See the Notes below for more details. |
| <i>FilesetId</i> | Pointer to a unique fileset identifier which specifies the fileset whose attributes are to be returned. If the <i>FilesetId</i> is NULL the <i>FilesetHandle</i> will be used. See the Notes below for more details. |
| <i>AttrBits</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be returned in the fileset attribute structure, <i>Attrs</i> , below. |
| <i>Attrs</i> | Pointer to a structure which is to contain the returned fileset data. |

Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

Error Conditions

The `ns_GetFilesetAttrs` routine is unsuccessful and the name server data base remains unchanged if any of the following are true:

| | |
|-----------------|--|
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The name server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EINVAL | Neither a <code>FilesetHandleP</code> or a <code>FilesetId</code> were supplied, or the <code>FilesetHandleP</code> and <code>FilesetId</code> do not point to the same fileset. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from or writing to the name server data base. |
| HPSS_ENOTREADY | The name server has not completed its initialization. |

See also

`ns_MkFileset`, `ns_SetFileAttrs`.

Clients

Client APIs, `insif`

Notes

If both the *FilesetHandle* and *FilesetId* are non-NULL the Name Server will insure that they both point to the same object. If they do not, an error will be returned. If both are NULL an error will be returned.

If a *FilesetHandle* is supplied performance will be slower because the Name Server will have to access the indicated object block.

2.1.7. ns_GetFilesetByNameOrId

Purpose

To return Global fileset record fields.

Syntax

```
#include <cns_interface.h>
```

```
signed32
```

```
ns_GetFilesetByNameOrId (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCreds,    /* IN */
    unsigned32            RequestId,     /* IN */
    unsigned32            Options,       /* IN */
    uchar                 *FilesetName,   /* IN/OUT */
    u_signed64            *FilesetId,    /* IN/OUT */
    uuid_t                *NameServerUUID, /* OUT */
    uuid_t                *GatewayUUID); /* OUT */
```

Description

This non-transactional procedure gets a Global fileset record and returns the fields of this record as parameters. *Option* controls the behavior of the function. See the Notes section for a description of the *Option* values. Note that the returned *GatewayUUID* (if any) is the UUID of the Gateway that manages the fileset. If there is no Gateway managing the fileset, the UUID returned for the Gateway will be NULL.

Parameters

| | |
|-----------------------|---|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the <i>hpss_ConnectHandle</i> that defines the connection context for this user. |
| <i>UserCreds</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestId</i> | The request identifier specified by the client. |
| <i>Options</i> | <i>Options</i> controls the behavior of of the function. See the Notes section below. |
| <i>FilesetName</i> | If the <i>Options</i> parameter indicates that we are to "get by name", this parameters points to a Fileset name. |
| <i>FilesetId</i> | If the <i>Options</i> parameter indicates that we are to "get by <i>FilesetId</i> ", this parameter contains the FilesetId. |
| <i>NameServerUUID</i> | The UUID of the Name Server that manages this Fileset. |
| <i>GatewayUUID</i> | The UUID of the Gateway that manages this fileset. Note that this value will be zero for all HPSS only filesets. |

Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

Error Conditions

The `ns_GetFilesetByNameOrId` routine is unsuccessful and the name server data base remains unchanged if any of the following are true:

| | |
|-----------------|--|
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The name server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EINVAL | The Options parameter contained an illegal value. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from or writing to the name server data base. |
| HPSS_ENOTREADY | The name server has not completed its initialization. |

See also

`ns_MkFileset`.

Clients

Client APIs, `insif`

Notes

The *Options* parameter controls the behavior of the routine. The permissible values for *Options* are:

| | |
|----------------------|---|
| NS_GET_BY_NAME | the function assumes that the client is supplying a <i>FilesetName</i> and the function will return the <i>FilesetId</i> , <i>NameServerUUID</i> , and the <i>GatewayUUID</i> . |
| NS_GET_BY_FILESET_ID | the function assumes the client is passing a <i>FilesetId</i> and the function returns the <i>FilesetName</i> , <i>NameServerUUID</i> and the <i>GatewayUUID</i> (if any). |

2.1.8. ns_GetName

Purpose

Get a path name for the specified bitfile.

Syntax

```
#include <hpssoid.h>
#include "cns_interface.h"
```

```
signed32
```

```
ns_GetName (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t        *UserCred,     /* IN */
    unsigned32             RequestID,     /* IN */
    hpssoid_t              *BitfileId,    /* IN */
    ns_ObjHandle_t         *ObjHandle,     /* IN */
    ns_ObjHandle_t         FilesetHandle, /* OUT */
    uchar                  *PathName,     /* OUT */
    gss_token_t            *AcsTicket);    /* OUT */
```

Description

This procedure gets one (of possibly many) path names for the bitfile identified by the specified *BitfileId*. In addition, it can be used to get the path name for the Name Server object identified by *ObjHandle*. It is an error to supply both a *BitfileId* and an *ObjHandle*.

Parameters

| | |
|----------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>BitfileId</i> | Pointer to the storage object ID for the bitfile whose path name is to be returned. |
| <i>ObjHandle</i> | Pointer to the Name Server object whose path name is to be returned. |
| <i>FilesetHandle</i> | Pointer to a Name Server object handle which identifies the root node of the fileset containing the <i>PathName</i> . The returned <i>PathName</i> is always relative to this object handle. |
| <i>PathName</i> | Pointer to a path name for the bitfile or Name Server object. |
| <i>AcsTicket</i> | Pointer to a ticket that contains the access rights (determined from the ACLs) to the bitfile for the user identified in <i>UserCred</i> . If this pointer is null on input, a ticket will not be returned. The return argument will be null if a connection to the BFS cannot be established or the path name refers to a directory or symbolic link. |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_GetName** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-------------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EINVAL | The path name identifies a "." or ".." directory. |
| HPSS_EIO | An internal error occurred while writing to the Name Server data base. |
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component in the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_Insert, **ns_MkDir**, **ns_Mk_Link**, **ns_MkSymLink**.

Clients

Storage System Manager, insif

Notes

The returned *PathName* will always begin with *./* and be relative to *FilesetHandle*

2.1.9. ns_Insert

Purpose

Insert a bitfile object into a directory.

Syntax

```
#include <hpssoid.h>
#include "cns_interface.h"
```

```
signed32
```

```
ns_Insert (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCred,     /* IN */
    unsigned32            RequestID,     /* IN */
    ns_ObjHandle_t        *Directory,     /* IN */
    uchar                  *PathName,     /* IN */
    unsigned32            DontBackUp,     /* IN */
    u_signed65            *FilesetId,     /* IN */
    ns_AttrBits_t         InAttrBits,     /* IN */
    ns_Attrs_t            *InAttrs,       /* IN */
    gss_token_t           *InAcsTicket,   /* IN */
    unsigned32            ReturnAttrsFlag, /* IN */
    ns_AttrBits_t         BitsForOutAttrs, /* IN */
    ns_Attrs_t            *OutAttrs,      /* OUT */
    ns_ObjHandle_t        *ObjHandle,     /* OUT */
    gss_token_t           *OutAcsTicket,  /* OUT */
    ns_RemainingPath_t    *RemainingPath); /* OUT */
```

Description

This procedure inserts a bitfile object which will be named by *PathName* into the specified directory. If return attributes are requested, the attributes specified in *BitsForOutAttrs* are returned in *OutAttrs*. If no *PathName* is supplied, an error is returned.

Parameters

| | |
|-------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory in which the bitfile is to be inserted. |
| <i>PathName</i> | The name of the bitfile that is to be inserted. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any “..” components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow “..” path components to back up past the |

| | |
|------------------------|---|
| | starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>FilesetId</i> | Pointer to the <i>FilesetId</i> of the fileset that the file is being inserted into. The purpose of this parameter is to insure the the new file is inserted into the correct fileset. The name server compares this <i>FilesetId</i> against the <i>FilesetId</i> of the file's parent directory. |
| <i>InAttrBits</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is provided in the attribute structure, <i>InAttrs</i> , below. |
| <i>InAttrs</i> | Pointer to the structure containing some values for the metadata that will be maintained by the Name Server for the specified bitfile. |
| <i>InAcsTicket</i> | Pointer to an access ticket containing the encrypted bitfile ID. This bitfile ID must match the bitfile ID specified in <i>InAttrs</i> . |
| <i>ReturnAttrsFlag</i> | Indicates if attributes are to be returned for the newly inserted bitfile. When the value of the flag is 1, the attributes specified by <i>BitsForOutAttrs</i> will be returned. If the value of the flag is 0, no attributes will be returned. |
| <i>BitsForOutAttrs</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be returned in <i>OutAttrs</i> . |
| <i>OutAttrs</i> | Pointer to a structure containing the attributes specified by <i>BitsForOutAttrs</i> for the newly inserted object. |
| <i>ObjHandle</i> | The Name Server handle that identifies the newly inserted bitfile. |
| <i>OutAcsTicket</i> | Pointer to a ticket that contains the access rights (determined from the ACLs) to the bitfile for the user identified in <i>UserCred</i> . If this pointer is null on input, a ticket will not be returned. The return argument will be null if a connection to the BFS cannot be established or the <i>PathName</i> refers to an object other than a file. |
| <i>RemainingPath</i> | Pointer to the structure containing returned information that is necessary to resolve the remainder of <i>PathName</i> . |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_Insert** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-------------|---|
| HPSS_EACCES | Search permission is denied on a component of the directory path or write permission is denied on the directory in which the bitfile is to be inserted. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |

| | |
|--------------------|--|
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn’t convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EBADF | The bitfile identifier is not to a valid file. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EEXIST | An object with the same path name already exists in the directory. |
| HPSS_EFAULT | An internal error occurred. |
| HPSS_EINCONSISTENT | The supplied <i>FilesetId</i> does not match the fileset being inserted into. |
| HPSS_EINVAL | The path name was “.”, “..” or NULL or a value in the attribute structure is invalid. |
| HPSS_EIO | An internal error occurred while writing to the Name Server data base. |
| HPSS_EMDATA | An inconsistency was encountered in the name server’s metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component of path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component of the path name does not exist. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_Delete, ns_Mkdir, ns_Mk_Link, ns_MkSymLink.

Clients

Client APIs, *insif*.

Notes

If both the UID and the GID have been supplied in the *InAttrs* structure these values will be assigned to the file. However if one or the other has been supplied, but not both, an error is returned. If neither the UID or GID have been supplied in the *InAttrs* structure, the UID is taken from the credentials, and the GID is taken from the parent directory. All other attribute values required to create the file will be obtained from *InAttrs*.

2.1.10. ns_MkDir

Purpose

Create a directory.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_MkDir (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCred,     /* IN */
    unsigned32            RequestID,     /* IN */
    ns_ObjHandle_t        *Directory,     /* IN */
    uchar                 *PathName,     /* IN */
    unsigned32            DontBackUp,     /* IN */
    u_signed64            *FilesetId,     /* IN */
    ns_AttrBits_t         InAttrBits,     /* IN */
    ns_Attrs_t            *InAttrs,       /* IN */
    unsigned32            ReturnAttrsFlag, /* IN */
    ns_AttrBits_t         BitsForOutAttrs, /* IN */
    ns_Attrs_t            *OutAttrs,      /* OUT */
    ns_ObjHandle_t        *ObjHandle,     /* OUT */
    ns_RemainingPath_t    *RemainingPath); /* OUT */
```

Description

This procedure is used to make a new directory identified by *PathName* in the specified directory with the attributes specified by *InAttrBits* with values found in *InAttrs*. If return attributes are requested, the attributes specified in *BitsForOutAttrs* are returned in *OutAttrs*.

Parameters

| | |
|-------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory in which the new directory is to be created. |
| <i>PathName</i> | Pointer to the path name of the new directory. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any ".." components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow ".." path components to back up past the starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |

| | |
|----------------------|--|
| <i>FilesetId</i> | Pointer to the <i>FilesetId</i> of the fileset that the directory is being inserted into. The purpose of this parameter is to insure the the new directory is inserted into the correct fileset. The name server compares this <i>FilesetId</i> against the FilesetId of the directory's parent directory. |
| <i>InAttrBits</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be obtained from the attribute structure, <i>InAttrs</i> , below. |
| <i>InAttrs</i> | Pointer to the structure containing some values for the metadata that will be maintained for the new directory. |
| ReturnAttrFlag | Indicates if attributes are to be returned for the newly created directory. When the value of the flag is 1, the attributes specified by <i>BitsForOutAttrs</i> will be returned. If the value of the flag is 0, no attributes will be returned. |
| BitsForOutAttrs | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be returned in <i>OutAttrs</i> . |
| OutAttrs | Pointer to a structure containing the attributes specified by <i>BitsForOutAttrs</i> for the newly created directory. |
| ObjHandle | The Name Server handle that identifies the newly created directory. |
| <i>RemainingPath</i> | Pointer to the structure containing returned information that is necessary to resolve the remainder of <i>PathName</i> . |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_MkDir** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-----------------|---|
| HPSS_EACCES | Search permission is denied on a component of the directory path or write permission is denied on the directory in which the new directory is to be made. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |

| | |
|--------------------|--|
| HPSS_EEXIST | An object with the same path name already exists in the directory. |
| HPSS_EINCONSISTENT | The supplied <i>FilesetId</i> does not match the fileset being inserted into. |
| HPSS_EINVAL | The path name was ".", ".." or NULL, or the value of a specified <i>InAttrs</i> is invalid. |
| HPSS_EIO | An internal error occurred while writing to the Name Server data base. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component of the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component of the path name does not exist. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_Delete, ns_Insert, ns_MkLink, ns_MkSymLink.

Clients

Client APIs, insif.

Notes

If both the UID and the GID have been supplied in the *InAttrs* structure these values will be assigned to the directory. However if one or the other has been supplied, but not both, an error is returned. If neither the UID or GID have been supplied in the *InAttrs* structure, the UID is taken from the credentials, and the GID is taken from the parent directory. All other attribute values required to create the directory will be obtained from *InAttrs*.

2.1.11. ns_MkFileset

Purpose

Create a fileset.

Syntax

```
#include <cns_interface.h>
```

```
signed32
```

```
ns_MkFileset (  
    trpc_handle_t          BindH,          /* IN */  
    hpss_connect_handle_t *ConnectH,     /* IN */  
    hsec_UserCred_t       *UserCreds,     /* IN */  
    unsigned32            RequestId,      /* IN */  
    ns_FilesetAttrBits_t  InFSAttrBits,   /* IN */  
    ns_FilesetAttrs_t     *InFSAttrs,     /* IN */  
    ns_AttrBits_t         InObjAttrBits,   /* IN */  
    ns_Attrs_t            *InObjAttrs,    /* IN */  
    ns_FilesetAttrBits_t  BitsForFSAttrs, /* IN */  
    ns_AttrBits_t         BitsForObjAttrs, /* IN */  
    ns_FilesetAttrs_t     *OutFSAttrs,    /* OUT */  
    ns_Attrs_t            *OutObjAttrs,   /* OUT */  
    ns_ObjHandle_t        *FilesetHandle); /* OUT */
```

Description

This procedure is used to create the root of a new fileset with the attributes specified by *InFSAttrBits* and *InObjAttrBits* using the values found in *InFSAttrs* and *InObjAttrs*.

Parameters

| | |
|-----------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the <i>hpss_ConnectHandle</i> that defines the connection context for this user. |
| <i>UserCreds</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestId</i> | The request identifier specified by the client. |
| <i>InFSAttrBits</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be obtained from <i>InFSAttrs</i> . |
| <i>InFSAttrs</i> | Pointer to the structure containing the values, if any, that are to be placed in the newly created fileset record. |
| <i>InObjAttrBits</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be obtained from <i>InObjAttrs</i> . |
| <i>InObjAttrs</i> | Pointer to the structure containing the values, if any, that are to be placed in the newly created fileset object block. |
| <i>BitsForFSAttrs</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be returned in <i>OutFSAttrs</i> . |

| | |
|------------------------|---|
| <i>BitsForObjAttrs</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be returned in <i>OutObjAttrs</i> . |
| <i>OutFSAttrs</i> | Pointer to a structure containing the attributes specified by <i>BitsForFSAttrs</i> for the newly created fileset. |
| <i>OutObjAttrs</i> | Pointer to a structure containing the attributes specified by <i>BitsForObjAttrs</i> for the newly created fileset object. |
| <i>FilesetHandle</i> | Pointer to Name Server object handle structure which will contain the object handle to the newly created fileset root object. |

Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

Error Conditions

The ns_MkFileset procedure is unsuccessful and the name server data base remains unchanged if any of the following are true:

| | |
|-----------------|---|
| HPSS_EACCES | The requestor does not have permission to make a fileset. Only the Root user is allowed to make filesets. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The name server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EEXISTS | The FilesetId was already being used. |
| HPSS_EINVAL | An input parameter was invalid. Check the log file for more details. |
| HPSS_EIO | An internal error occurred while writing to the name server data base. |
| HPSS_ENOTREADY | The name server has not completed its initialization. |

See also

ns_DeleteFileset

Clients

Client APIs, insif

Notes

The root node of a fileset is created. This object will not be a part of any existing Name Server directory heirarchy. It will be a stand-alone directory which is separate from the Name Server's root.

If both the UID and the GID have been supplied in the *InObjAttrs* structure these values will be assigned to the fileset root. However if one or the other has been supplied, but not both, an error is returned. If neither the UID or GID have been supplied in the *InObjAttrs* structure, the UID is taken from the credentials, and the GID is taken from the parent directory. All other object attribute values required to create the fileset root will be obtained from *InObjAttrs*.

2.1.12. ns_MkJunction

Purpose

Create a junction point.

Syntax

```
#include <cns_interface.h>
```

```
signed32
```

```
ns_MkJunction (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCreds,    /* IN */
    unsigned32            RequestId,     /* IN */
    ns_ObjHandle_t        *Directory,    /* IN */
    uchar                 *PathName,     /* IN */
    unsigned32            DontBackUp,    /* IN */
    u_signed64            *FilesetId,    /* IN */
    ns_ObjHandle_t        *FilesetHandle, /* IN */
    ns_AttrBits_t         AttrBits,      /* IN */
    ns_Attrs_t            *Attrs,        /* IN */
    ns_ObjHandle_t        *JunctionHandle, /* OUT */
    ns_RemainingPath_t    *RemainingPath); /* OUT */
```

Description

This procedure is used to make a directory junction identified by *PathName* in the specified directory using the attributes specified by *AttrBits* with values found in *Attrs*. The resulting junction will point to the directory specified by *FilesetHandle*.

Parameters

| | |
|-------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the <i>hpss_ConnectHandle</i> that defines the connection context for this user. |
| <i>UserCreds</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestId</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory in which the new junction is to be created. |
| <i>PathName</i> | Pointer to the path name of the new junction. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any “..” components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow “..” path components to back up past the starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>FilesetId</i> | Pointer to the <i>FilesetId</i> of the fileset that the junction is being inserted into. The purpose of this parameter is to insure the the |

new junction is inserted into the correct fileset. The name server compares this *FilesetId* against the *FilesetId* of the junction's parent directory.

| | |
|-----------------------|---|
| <i>FilesetHandle</i> | A name server object handle that points to the root node of a fileset. The root node of a fileset is a directory. |
| <i>AttrBits</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be obtained from the attribute structure, <i>Attrs</i> , below. |
| <i>Attrs</i> | Pointer to the structure containing the values for some metadata fields that will be maintained for the new junction. |
| <i>JunctionHandle</i> | The name server object handle that identifies the newly created junction. |
| <i>RemainingPath</i> | Pointer to the structure containing returned information that is necessary to resolve the remainder of <i>PathName</i> . |

Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

Error Conditions

The *ns_MkJunction* procedure is unsuccessful and the name server data base remains unchanged if any of the following are true:

| | |
|-----------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The name server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EEXIST | An object with the same path name already exists in the directory. |
| HPSS_EINVAL | The path name was “.”, “..” or NULL, or the value of a specified <i>InAttr</i> is invalid. |
| HPSS_EIO | An internal error occurred while writing to the name server data base. |

| | |
|-------------------|---|
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component of the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | If the <i>PathName</i> is provided, a component of the <i>PathName</i> does not exist or the <i>PathName</i> argument points to an empty string. If only an object handle is provided, the object does not exist. |
| HPSS_ENOTDIR | The Directory object handle is not to a valid directory. |
| HPSS_ENOTREADY | The name server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the Directory is incorrect. |

See also

ns_MkFileset, ns_DeleteFileset.

Clients

Client APIs, insif

Notes

If both the UID and the GID have been supplied in the *Attrs* structure these values will be assigned to the junction. However if one or the other has been supplied, but not both, an error is returned. If neither the UID or GID have been supplied in the *Attrs* structure, the UID is taken from the credentials, and the GID is taken from the parent directory. All other attribute values required to create the junction will be obtained from *Attrs*.

Junction points can only "point" to directories, anything else is an error. The supplied *FilesetHandleP* must be of type Directory.

Only the Root user or a trusted user with DCE Write permission can create Junctions.

2.1.13. ns_MkLink

Purpose

Create a (hard) link to a file.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_MkLink (  
    trpc_handle_t          BindH,          /* IN */  
    hpss_connect_handle_t *ConnectH,     /* IN */  
    hsec_UserCred_t       *UserCred,     /* IN */  
    unsigned32            RequestID,     /* IN */  
    ns_ObjHandle_t       *Directory,     /* IN */  
    u_char                *PathName,     /* IN */  
    unsigned43            DontBackUp,     /* IN */  
    u_signed64            *FilesetId,     /* IN */  
    ns_ObjHandle_t       *BitfileObjId,  /* IN */  
    ns_ObjHandle_t       *ObjHandle,     /* OUT */  
    ns_RemainingPath_t   *RemainingPath); /* OUT */
```

Description

This procedure is used to make a (hard) link entry identified by *PathName* in the specified directory. This entry will be an alternate path (hard link) to the specified bitfile. The file being linked to must be in the same fileset as the hard link.

Parameters

| | |
|-------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory in which the link is to be made. |
| <i>PathName</i> | Pointer to the path name for the link. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any ".." components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow ".." path components to back up past the starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>FilesetId</i> | Pointer to the <i>FilesetId</i> of the fileset that the link is being inserted into. The purpose of this parameter is to insure the the new link is inserted into the correct fileset. The name server compares this <i>FilesetId</i> against the <i>FilesetId</i> of the link's parent directory. |

| | |
|----------------------|--|
| <i>BitFileObjId</i> | The Name Server handle to the bitfile that is to be linked. |
| <i>ObjHandle</i> | The Name Server handle that identifies the newly created hard link. |
| <i>RemainingPath</i> | Pointer to the structure containing returned information that is necessary to resolve the remainder of <i>PathName</i> . |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_MkLink** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|--------------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path or write permission is denied on the directory in which the link is to be made. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn’t convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EEXIST | An object with the same path name already exists in the directory. |
| HPSS_EINCONSISTENT | The supplied <i>FilesetId</i> does not match the fileset being inserted into. |
| HPSS_EINVAL | The path name was “.”, “..” or NULL or a value in the attribute structure is invalid. |
| HPSS_EIO | An internal error occurred while writing to the Name Server data base. |
| HPSS_EMDATA | An inconsistency was encountered in the name server’s metadata. |
| HPSS_EMLINK | The maximum number of links, 32767, already exists. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |

| | |
|-------------------|--|
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component of the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component of the PathName does not exist or the PathName argument points to an empty string. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |
| HPSS_EXDEV. | The link named by <i>PathName</i> and the file named by <i>BitFileObjId</i> are on different filests. |

See also

ns_Delete, ns_Insert, ns_MkDir, ns_MkSymLink.

Clients

Client APIs, *insif*.

Notes

The UID associated with the newly linked bitfile will be that of the UID found in the credentials structure. The GID associated with the newly linked bitfile will be that of the GID associated with the parent directory.

2.1.14. ns_MkSymLink

Purpose

Make a symbolic link.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_MkSymLink (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCred,     /* IN */
    unsigned32            RequestID,     /* IN */
    ns_ObjHandle_t        *Directory,    /* IN */
    uchar                  *PathName,    /* IN */
    unsigned32            DontBackUp,    /* IN */
    u_signed64             *FilesetId,    /* IN */
    uchar                  *LinkText,    /* IN */
    ns_ObjHandle_t        *ObjHandle,    /* OUT */
    ns_RemainingPath_t    *RemainingPath); /* OUT */
```

Description

This transactional procedure is used to make a symbolic link identified by *PathName* in the specified directory. The text stored for the symbolic link is pointed to by *LinkText*.

Parameters

| | |
|-------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory in which the symbolic link is to be made. |
| <i>PathName</i> | Pointer to the path name for the symbolic link. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any ".." components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow ".." path components to back up past the starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>FilesetId</i> | Pointer to the <i>FilesetId</i> of the fileset that the symbolic link is being inserted into. The purpose of this parameter is to insure the the new symbolic link is inserted into the correct fileset. The name server compares this <i>FilesetId</i> against the <i>FilesetId</i> of the symbolic link's parent directory. |

| | |
|------------------|--|
| <i>LinkText</i> | Pointer to the data that will be stored for the symbolic link. |
| <i>ObjHandle</i> | The Name Server handle that identifies the newly created symbolic link. |
| RemainingPathP | Pointer to the structure containing returned information that is necessary to resolve the remainder of <i>PathName</i> . |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_MkSymLink** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|--------------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path or write permission is denied on the directory in which the symbolic link is to be made. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn’t convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EEXIST | An object with the same path name already exists in the directory. |
| HPSS_EINCONSISTENT | The supplied <i>FilesetId</i> does not match the fileset being inserted into. |
| HPSS_EINVAL | The path name was “.”, “..” or NULL. |
| HPSS_EIO | An internal error occurred while writing to the Name Server data base. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |
| HPSS_ENAMETOOLONG | The length of the path name or the link text exceeds HPSS_MAX_PATH_NAME or the length of a component in the path name exceeds HPSS_MAX_FILE_NAME. |

| | |
|----------------|--|
| HPSS_ENOENT | A component of the PathName does not exist or the PathName argument points to an empty string. HPSS_ENOTDIR The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_Delete, ns_Insert, ns_MkDir, ns_MkLink.

Clients

Client APIs, insif.

Notes

The UID associated with the newly created symbolic link will be that of the UID found in the credentials structure. The GID associated with the newly created symbolic link will be that of the GID associated with the parent directory.

2.1.15. ns_NSGetAttrs

Purpose

Get the configurable data of the Name Server.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_NSGetAttrs (  
    trpc_handle_t          BindH,          /* IN */  
    hpss_connect_handle_t *ConnectH,     /* IN */  
    hsec_UserCred_t       *UserCred,     /* IN */  
    unsigned32            RequestID,     /* IN */  
    ns_SpecificConfig_t   *ConfigData); /* OUT*/
```

Description

Non-transactional procedure used to return the Name Server's configurable data in *ConfigData*.

Parameters

| | |
|-------------------|---|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>ConfigData</i> | Pointer to the structure which will contain the values from the specific configurable data. |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The *ns_NSGetAttrs* procedure is unsuccessful and the Name Server's configurable data is not returned if any of the following are true:

| | |
|-----------------|--|
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |

See also

ns_NSSetAttrs.

Clients

Storage System Manager.

Notes

The in-memory copy of the configuration data is returned by this API. The configuration data is read once during initialization from the Name Server's specific configuration file. Any changes made to the file after the Name Server's initialization will not be visible until the Name Server is restarted.

2.1.16. ns_NSSetAttrs

Purpose

Set the modifiable fields of the in-memory copy of the Name Server's configuration data.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_NSSetAttrs (  
    trpc_handle_t          BindH,          /* IN */  
    hpss_connect_handle_t *ConnectH,     /* IN */  
    hsec_UserCred_t        *UserCred,     /* IN */  
    unsigned32             RequestID,     /* IN */  
    ns_ConfigBits_t        InConfigBits,  /* IN */  
    ns_SpecificConfig_t    *InConfigData, /* IN */  
    ns_ConfigBits_t        *OutConfigBits, /* OUT */  
    ns_SpecificConfig_t    *OutConfigData); /* OUT */
```

Description

Set the fields of the configuration data specified by *InConfigBits* to the values in *InConfigData*.

Parameters

| | |
|----------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>InConfigBits</i> | A bit vector in which the appropriate bit is set (on) for each field in the configuration data provided in <i>InConfigData</i> . |
| <i>InConfigData</i> | Pointer to the structure containing the values that the configuration data is to be set to. |
| <i>OutConfigBits</i> | A bit vector in which the appropriate bit is set (on) for each of the fields in the configuration data that were set. |
| <i>OutConfigData</i> | Pointer to the structure containing the values that the configurable data was set to. |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_NSSetAttrs** procedure is unsuccessful and the Name Server's configuration data remains unchanged if any of the following are true:

| | |
|---------------|---|
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
|---------------|---|

| | |
|-----------------|--|
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |

See also

ns_NSGetAttrs.

Clients

Storage System Manager.

Notes

Only the in-memory copy of the name server's configuration data will be changed. The data stored in the file can be modified through the metadata manager utilities. The only fields of the configuration data that can be set are the maximum path components, maximum byte size of the readdir buffer and the default file and directory permissions.

2.1.17. ns_ReadDir

Purpose

Return a list of directory entries.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_ReadDir (  
    trpc_handle_t          BindH,          /* IN */  
    hpss_connect_handle_t *ConnectH,     /* IN */  
    hsec_UserCred_t        *UserCred,     /* IN */  
    unsigned32             RequestID,     /* IN */  
    ns_ObjHandle_t        *Directory,     /* IN */  
    unsigned32             Offset,        /* IN */  
    unsigned32             BuffSize,     /* IN */  
    unsigned32             ReturnAttrsFlag, /* IN */  
    ns_AttrBits_t          BitsForOutAttrs, /* IN */  
    unsigned32             *EndOfDir,     /* OUT */  
    ns_DirEntry_t         **Entries);    /* OUT */
```

Description

Return a list of directory entries starting at the location indicated by *Offset*, the continuation cookie. Entries will be returned until the addition of another would cause the total to exceed the specified buffer size, or until the end of the directory is reached. The value of *EndOfDir* will be 1 if an attempt is made to read past the last entry. If the attributes for the entries are requested those attributes specified in *BitsForOutAttrs* will be returned for each entry in the directory.

Parameters

| | |
|------------------------|---|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory whose contents are to be read. |
| <i>Offset</i> | Pointer to the entry in the directory where the listing is to begin. The <i>Offset</i> is also known as the continuation cookie. |
| <i>BuffSize</i> | The size in bytes of the buffer that the client has to hold the entries that will be read. |
| <i>ReturnAttrsFlag</i> | Indicates if attributes are to be returned for each directory entry. When this value is 1, the attributes specified by <i>BitsForOutAttrs</i> will be returned. If the value of this flag is 0, no attributes will be returned. |

| | |
|------------------------|---|
| <i>BitsForOutAttrs</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be returned in the attribute structure associated with each directory entry. |
| <i>EndOfDir</i> | A pointer to a flag that is set to 1 whenever the Name Server reaches the end of the directory. |
| <i>Entries</i> | Pointer to the list of directory entries that are returned. Each entry in the list consists of the object's name, Name Server object handle, a continuation cookie, and any requested attributes. |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_ReadDir** procedure is unsuccessful and no *Entries* are returned if any of the following are true:

| | |
|-----------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path or read permission is denied on the directory that is to be read. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from or writing to the Name Server data base. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ERANGE | The buffer used by the client to receive the list of directory entries is too small to hold a complete entry. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_GetAttr, ns_ReadLink.

Clients

Client APIs, *insif*.

Notes

None.

2.1.18. ns_ReadFilesetAttrs

Purpose

Return an array of fileset entries.

Syntax

```
#include <cns_interface.h>
```

```
signed32
```

```
ns_ReadFilesetAttrs (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCreds,     /* IN */
    unsigned32            RequestId,      /* IN */
    u_signed64            Offset,         /* IN */
    unsigned32            HowMany,        /* IN */
    unsigned32            *EndOfFSEntries, /* OUT */
    ns_FilesetAttrsConfArray_t);          /* OUT */
```

Description

Return an array of fileset entries starting at the location indicated by *Offset*, the continuation cookie. Fileset entries will be returned until all of the fileset entries have been returned or until *HowMany* entries have been placed in the *FilesetEntries* array. The value of *EndOfFSEntries* will be set to 1 whenever there are no more *FilesetAttr* records to return.

Parameters

| | |
|-----------------------|---|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the <i>hpss_ConnectHandle</i> that defines the connection context for this user. |
| <i>UserCreds</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestId</i> | The request identifier specified by the client. |
| <i>Offset</i> | Pointer to the fileset entry where the listing is to begin. The <i>Offset</i> is also known as the Continuation cookie. <i>Offset</i> should have a value of zero to begin reading at the beginning of the fileset entries. |
| <i>HowMany</i> | The maximum number of fileset entries the client wishes to have returned. The name server will return the minimum of <i>HowMany</i> and <i>NS_FS_MAX_ENTRIES_TO_RETURN</i> . |
| <i>EndOfFSEntries</i> | A pointer to a flag that is set to 1 when there are no more <i>FilesetAttr</i> records to return. |
| <i>FilesetEntries</i> | Pointer to the conformant array of fileset entries that are returned. Each element of the array contains the fileset name, <i>FilesetId</i> , a continuation cookie, the fileset state, and the fileset type. |

Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

Error Conditions

The `ns_ReadFilesetAttrs` procedure is unsuccessful and no `FSEntries` are returned if any of the following are true:

| | |
|------------------------------|--|
| <code>HPSS_EAGAIN</code> | Resources are temporarily unavailable. |
| <code>HPSS_EBADCONN</code> | The supplied connection context is not formatted as expected. |
| <code>HPSS_EBADCREDS</code> | The name server couldn't convert the HPSS credentials into a form suitable for internal use. |
| <code>HPSS_EMDATA</code> | An inconsistency was encountered in the name server's metadata. |
| <code>HPSS_EENTERCONN</code> | An error was encountered while attempting to enter the connection. |
| <code>HPSS_EIO</code> | An internal error occurred while reading from the name server data base. |
| <code>HPSS_ENOTREADY</code> | The name server has not completed its initialization. |

See also

`ns_DeleteFileset`, `ns_GetFilesetAttrs`, and `ns_MkFileset`.

Clients

Client APIs, `insif`

Notes

None.

2.1.19. ns_ReadGlobalFilesets

Purpose

Return an array of global fileset entries.

Syntax

```
#include <cns_interface.h>
```

```
signed32
```

```
ns_ReadGlobalFilesets(
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCreds,    /* IN */
    unsigned32            RequestId,     /* IN */
    unsigned32            WhichUUID,     /* IN */
    uuid_t                *UUID,        /* IN */
    u_signed64            Offset,        /* IN */
    unsigned32            HowMany,       /* IN */
    unsigned32            *EndOfEntries, /* OUT */
    ns_GFilesetConfArray_t **GFilesetEntries); /* OUT */
```

Description

Return an array of global fileset entries from the global fileset metadata file. *WhichUUID* indicates whether the returned list is for all of the filesets, or is for the filesets managed by a particular Name Server or Gateway. See below for more details about *WhichUUID*. The starting location in the global fileset array is indicated by *Offset*, the continuation cookie. Entries will be returned until all of the global fileset entries have been returned or until *HowMany* entries have been placed in the *GFilesetEntriesPP* array. The value of *EndOfEntries* will be set to 1 when there are no more global entries to return.

Parameters

| | |
|------------------|---|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the <i>hpss_ConnectHandle</i> that defines the connection context for this user. |
| <i>UserCreds</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestId</i> | The request identifier specified by the client. |
| <i>WhichUUID</i> | Determines which entries are to be read from the global fileset array. See the Notes below for details. |
| <i>UUID</i> | Depending on the value of <i>WhichUUID</i> this parameter may point to either a Gateway UUID or a Name Server UUID. |
| <i>Offset</i> | Pointer to the global fileset entry where the listing is to begin. The <i>Offset</i> is also known as the Continuation cookie. <i>Offset</i> should have a value of zero to begin reading at the beginning of the global fileset entries. |

| | |
|------------------------|--|
| <i>HowMany</i> | The maximum number of fileset entries the client wishes to have returned. The name server will return the minimum of <i>HowMany</i> and NS_FS_MAX_ENTRIES_TO_RETURN. |
| <i>EndOfEntries</i> | A pointer to a flag that is set to 1 whenever there are no more entries to read. |
| <i>GFilesetEntries</i> | Pointer to the conformant array of fileset entries that are returned. Each element of the array contains the fileset name, FilesetId, a continuation cookie, the UUID of the gateway that manages this fileset, and the UUID of the Name Server that manages this fileset. |

Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

Error Conditions

The ns_ReadGlobalFilesets procedure is unsuccessful and no Entries are returned if any of the following are true:

| | |
|-----------------|--|
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The name server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from the name server data base. |
| HPSS_ENOTREADY | The name server has not completed its initialization. |

See also

ns_ReadFilesets.

Clients

Client APIs, insif

Notes

WhichUUID may contain any of the following values:

| | |
|-------------------------|--|
| NS_READ_G_FS_ALL | All of the entries in the global filesets array will be read. |
| NS_READ_G_FS_BY_GW_UUID | The UUID parameter points to the UUID of a particular Gateway. Only the entries having this Gateway UUID will be read. |

NS_READ_G_FS_BY_NS_UUID

The UUID parameter points to the UUID of a particular Name Server. Only the entries having this Name Server UUID will be read.

2.1.20. ns_ReadJunctionPathNames

Purpose

Return an array of Junction path names.

Syntax

```
#include <cns_interface.h>
```

```
signed32
```

```
ns_ReadJunctionPathNames(
```

```
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,      /* IN */
    hsec_UserCred_t        *UserCreds,     /* IN */
    unsigned32             RequestId,      /* IN */
    u_signed64             Offset,         /* IN */
    unsigned32             HowMany,        /* IN */
    unsigned32             *EndOfEntries,  /* OUT */
    ns_JunctionPathConfArray_t **JunctionEntries); /* OUT */
```

Description

Return an array of path names to all of the Junctions managed by this Name Server. The starting location in the *JunctionEntries* array is indicated by *Offset*, the continuation cookie. Junction entries will be returned until all of the junction entries have been returned or until *HowMany* entries have been placed in the *JunctionEntries* array. The value of *EndOfEntries* will be set to 1 when there are no more junction entries to return.

Parameters

| | |
|------------------------|---|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the <i>hpss_ConnectHandle</i> that defines the connection context for this user. |
| <i>UserCreds</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestId</i> | The request identifier specified by the client. |
| <i>Offset</i> | Pointer to the junction entry where the listing is to begin. The <i>Offset</i> is also known as the Continuation cookie. <i>Offset</i> should have a value of zero to begin reading at the beginning of the junction entries. |
| <i>HowMany</i> | The maximum number of junction entries the client wishes to have returned. The name server will return the minimum of <i>HowMany</i> and <i>NS_FS_MAX_ENTRIES_TO_RETURN</i> . |
| <i>EndOfEntries</i> | A pointer to a flag that is set to 1 whenever there are no more entries to read. |
| <i>JunctionEntries</i> | Pointer to the conformant array of junction entries that are returned. Each element of the array contains a fileset handle, junction handle, <i>Offset</i> , and the path name to the Junction. |

Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

Error Conditions

The `ns_ReadJunctionPathNames` procedure is unsuccessful and no entries are returned if any of the following are true:

| | |
|-----------------|--|
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The name server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from the name server data base. |
| HPSS_ENOTREADY | The name server has not completed its initialization. |

See also

`ns_ReadFilesets`.

Clients

Client APIs, `insif`

Notes

none

2.1.21. ns_ReadLink

Purpose

Read the data associated with a symbolic link.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_ReadLink (  
    trpc_handle_t          BindH,          /* IN */  
    hpss_connect_handle_t *ConnectH,     /* IN */  
    hsec_UserCred_t        *UserCred,     /* IN */  
    unsigned32             RequestID,     /* IN */  
    ns_ObjHandle_t        *Directory,     /* IN */  
    uchar                  *PathName,     /* IN */  
    unsigned32             DontBackUp,    /* IN */  
    uchar                  *LinkText,     /* OUT */  
    ns_RemainingPath_t    *RemainingPath); /* OUT */
```

Description

This procedure is used to return the data associated with the symbolic link identified by *PathName* in the specified *Directory*. If no *PathName* is supplied, the symbolic link identified by *Directory* will be returned.

Parameters

| | |
|----------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory which contains the symbolic link. |
| <i>PathName</i> | Pointer to the path name for the symbolic link. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any ".." components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow ".." path components to back up past the starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>LinkText</i> | Pointer to the uninterpreted data that is to be returned. |
| <i>RemainingPath</i> | Pointer to the structure containing returned information that is necessary to resolve the remainder of <i>PathName</i> . |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_ReadLink** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-------------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path or read permission is denied on the symbolic link that is to be read. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn’t convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from the name server data base. |
| HPSS_EMDATA | An inconsistency was encountered in the name server’s metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component in the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component of the path name does not exist. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_GetAttr, ns_MkSymLink.

Clients

Client APIs, *insif*.

Notes

None.

2.1.22. ns_Rename

Purpose

Rename a name space object.

Syntax

```
#include "cns_interface.h"
signed32
ns_Rename (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t        *UserCred,     /* IN */
    unsigned32             RequestID,     /* IN */
    ns_ObjHandle_t         *CurrentDir,    /* IN */
    uchar                  *CurrentPath,   /* IN */
    ns_ObjHandle_t         *NewDir,        /* IN */
    uchar                  *NewPath,       /* IN */
    unsigned32             DontBackUp,     /* IN */
    ns_RemainingPath_t     *CurrentRemainPath, /* OUT */
    ns_RemainingPath_t     *NewReaminPath); /* OUT */
```

Description

Rename a Name Server object to the specified new name. The new name may not previously exist. The renamed entry specified by *NewDir* and *NewPath* must reside in the same fileset as the current entry named by *CurrentDir* and *CurrentPath*.

Parameters

| | |
|--------------------------|---|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>CurrentDir</i> | Pointer to the directory where the object currently resides. |
| <i>CurrentPath</i> | Pointer to the current path name of the object. |
| <i>NewDir</i> | Pointer to directory where the object is to be moved. |
| <i>NewPath</i> | Pointer to the new path name of the object. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 the name server will stop processing the request if any ".." components in either the <i>CurrentPath</i> or the <i>NewPath</i> result in backing up past the starting directory. If the value is 0 then the name server will allow ".." path components in either the <i>CurrentPath</i> or the <i>NewPath</i> to back up past the starting directory. |
| <i>CurrentRemainPath</i> | Pointer to the structure containing the information necessary to resolve the remainder of the <i>CurrentPath</i> . |

NewRemainPath Pointer to the structure containing the information necessary to resolve the remainder of the *NewPath*.

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The ns_Rename procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-------------------|--|
| HPSS_EACCES | Search permission is denied on a component of either the current or new path name or write permission is denied on either the current or new directory. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A <i>CurrentPath</i> or a <i>NewPath</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn’t convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EEXIST | An object specified by the new path name already exists in the new directory. |
| HPSS_EINVAL | The object to be renamed (moved) is a parent directory of the current directory. This error is also returned if either the current or new path name is “.”, “..” or NULL. |
| HPSS_EIO | An internal error occurred while reading from or writing to the Name Server data base. |
| HPSS_ENAMETOOLONG | The length of either the current or new path name exceeds HPSS_MAX_PATH_NAME or the length of a component in either the current or new path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component of either the current or new path does not exist or the argument to the new path name points to an empty string. |
| HPSS_ENOTDIR | Either the current or new object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

HPSS_EXDEV.

The entry named by *CurrentDir* and *CurrentPath* and the entry named by *NewDir* and *NewPath* are on different filesets.

See also

None.

Clients

Client APIs, insif.

Notes

The POSIX standards state that, if the new path name already exists in the new directory, it is removed. The HPSS design requires that the error EEXIST be returned if this situation occurs. An explicit delete of any existing entry with new name must be done by the client before the rename can be successful.

The resulting new name must exist in the same fileset as the Current name.

2.1.23. ns_ServerGetAttrs**Purpose**

Get the Name Server state data.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_ServerGetAttrs (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCred,     /* IN */
    unsigned32            RequestID,     /* IN */
    hpss_server_attr_t    *StateData);   /* OUT */
```

Description

Get the Name Server's global state data.

Parameters

| | |
|------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>StateData</i> | Pointer to the structure containing the Name Server's state. |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_ServerGetAttrs** procedure is unsuccessful and the Name Server state data is not returned if any of the following are true:

| | |
|-----------------|--|
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while processing the request. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |

See also

ns_ServerSetAttrs.

Clients

Storage System Manager, insif.

Notes

None.

2.1.24. ns_ServerSetAttrs

Purpose

Set the Name Server state data.

Syntax

```
#include "cns_interface.h"

signed32
ns_ServerSetAttrs (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCred,     /* IN */
    unsigned32            RequestID,     /* IN */
    u_signed64             StateBits,     /* IN */
    hpss_server_attr_t    *StateData,    /* IN */
    u_signed64             *OutStateBits, /* OUT */
    hpss_server_attr_t    *OutStateData); /* OUT */
```

Description

Set the Name Server's global state data specified by *StateBits* to the values in *StateData*.

Parameters

| | |
|---------------------|---|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>StateBits</i> | A bit vector indicating which fields of the Name Server state data are to be set. |
| <i>StateData</i> | Pointer to the structure containing the data to set the Name Server's state to. |
| <i>OutStateBits</i> | A bit vector indicating which fields of the Name Server state data were set. |
| <i>OutStateData</i> | Pointer to the structure containing the state data that the Name Server was set to. |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_ServerSetAttrs** procedure is unsuccessful and the Name Server state data is not set if any of the following are true:

| | |
|--------------------|---|
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while processing the request. |
| HPSS_EPERM | Only SSM is allowed to set the Name Server's global state data. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ENOTSUPPORTED | The Name Server does not currently support the requested operation. |

See also

ns_ServerGetAttrs.

Clients

Storage System Manager, insif.

Notes

None.

2.1.25. ns_SetACL

Purpose

Set the ACL of the specified Name Server object.

Syntax

```
#include "cns_interface.h"

signed32
ns_SetACL (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCred,     /* IN */
    unsigned32            RequestID,     /* IN */
    ns_ObjHandle_t        *Directory,    /* IN */
    uchar                 *PathName,     /* IN */
    unsigned32            DontBackUp,    /* IN */
    ns_ACLConfArray_t     *NewACLEntries, /* IN */
    ns_RemainingPath_t    *RemainingPath); /* OUT */
```

Description

Set the ACL of the object identified by *PathName* to that specified by the new ACL. If no *PathName* is supplied, the ACLs are set on the object identified by *Directory*.

Parameters

| | |
|----------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory containing the object whose ACL is to be set. |
| <i>PathName</i> | Pointer to the path name of the object whose ACL is to be set. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any ".." components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow ".." path components to back up past the starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>NewACLEntries</i> | Pointer to the array of new ACL entries. If <i>NewACLEntries</i> is null, all ACL entries for the object will be deleted. |
| <i>RemainingPath</i> | Pointer to the structure containing returned information that is necessary to resolve the remainder of <i>PathName</i> . |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_SetACL** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-------------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path, or the owner does not have write permission, or others do not have control permission. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn’t convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from or writing to the Name Server data base. |
| HPSS_EMDATA | An inconsistency was encountered in the name server’s metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component in the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component in the path name does not exist. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_DeleteACL, ns_GetACL, ns_UpdateACL.

Clients

Client APIs, *insif*.

Notes

None.

2.1.26. ns_SetAttrs

Purpose

Set the attributes of the specified Name Server object.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_SetAttrs (
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCred,     /* IN */
    unsigned32            RequestID,     /* IN */
    ns_ObjHandle_t       *Directory,     /* IN */
    uchar                 *PathName,     /* IN */
    unsigned32            DontBackUp,     /* IN */
    ns_AttrBits_t        InAttrBits,     /* IN */
    ns_Attrs_t           *InAttrs,       /* IN */
    gss_token_t          *InAcsTicket,   /* IN */
    unsigned32            ReturnAttrsFlag, /* IN */
    ns_AttrBits_t        BitsForOutAttrs, /* IN */
    ns_Attrs_t           *OutAttrs,      /* OUT */
    gss_token_t          *OutAcsTicket, /* OUT */
    ns_RemainingPath_t   *RemainingPath); /* OUT */
```

Description

Set the metadata fields of the object identified by *Directory* and *PathName*. The specific attributes to be set are identified by *InAttrBits* and the values these fields are to be set to are found in *InAttrs*. If return attributes are requested, the attributes specified in *BitsForOutAttrs* are returned in *OutAttrs*. If no *PathName* is supplied, the attributes are set on the object identified by *Directory*.

Parameters

| | |
|-------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory containing the object whose attributes are to be set. |
| <i>PathName</i> | Pointer to the path name of the object whose attributes are to be set. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any ".." components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow ".." path components to back up past the |

| | |
|------------------------|---|
| | starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>InAttrBits</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is provided in the attribute structure, <i>InAttrs</i> , below. |
| <i>InAttrs</i> | Pointer to the structure containing the values that the object's fields are to be set to. |
| <i>InAcsTicket</i> | Pointer to an access ticket containing the encrypted bitfile ID. If the bitfile ID is to be set, this bitfile ID must match the bitfile ID specified in <i>InAttrs</i> . |
| <i>ReturnAttrFlag</i> | Indicates if the newly set attributes are to be returned. When the value of the flag is 1, the attributes specified by <i>BitsForOutAttrs</i> will be returned. If the value of the flag is 0, no attributes will be returned. |
| <i>BitsForOutAttrs</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be returned in <i>OutAttrs</i> . |
| <i>OutAttrs</i> | Pointer to a structure containing the attributes specified by <i>BitsForOutAttrs</i> for the object. |
| <i>OutAcsTicket</i> | Pointer to a ticket that contains the access rights (determined from the ACLs) to the bitfile for the user identified in <i>UserCred</i> . If this pointer is null on input, a ticket will not be returned. The return argument will be null if a connection to the BFS cannot be established or the path name refers to an object other than a file. |
| <i>RemainingPathP</i> | Pointer to the structure containing returned information that is necessary to resolve the remainder of <i>PathName</i> . |

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_SetAttr** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|----------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory name or write permission is denied on the object whose attributes are to be set. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to 'true'. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |

| | |
|-------------------|---|
| HPSS_EBADCREDS | The Name Server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EBADF | The bitfile identifier is not to a valid file. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EINVAL | The path name identifies a "." or ".." directory. |
| HPSS_EIO | An internal error occurred while reading from or writing to the Name Server data base. |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component in the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component of the directory path does not exist. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_EPERM | The UID in the credentials is not super-user or the owner of the object or the owner is not a member of the new group. This error can occur when updating the mode, times, UID or GID fields. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_GetAttr.

Clients

Client APIs, insif.

Notes

Valid bits for the *InAttrBits* are defined in the data definition section. Certain attributes associated with a bit file, such as size and bits for *OutAttrs*, time last accessed and time last modified must be changed through the BFS. To change the UID of the object the credentials must belong to either super-user or to the owner of the object. To change the GID of the object the credentials must belong to either super-user or to the owner of the object and the owner must be a member of the new group. To change the "accessed" or "modified" times of the object the credentials must belong to either the super-user or the owner of the object. Any user with write access to the object may update the "accessed" or "modified" times to the current time by setting the appropriate bit in the bit map and supplying a null value for the time.

Fileset attributes can only be changed using the ns_SetFilesetAttrs API.

2.1.27. ns_SetFilesetAttrs

Purpose

Set the attributes of the specified name server fileset.

Syntax

```
#include <cns_interface.h>
```

```
signed32
```

```
ns_SetFilesetAttrs (  
    trpc_handle_t          BindH,          /* IN */  
    hpss_connect_handle_t *ConnectH,     /* IN */  
    hsec_UserCred_t       *UserCreds,    /* IN */  
    unsigned32            RequestId,     /* IN */  
    ns_ObjHandle_t        *FilesetHandle, /* IN */  
    u_signed64            *FilesetId,    /* IN */  
    ns_FilesetAttrBits_t  InAttrBits,    /* IN */  
    ns_FilesetAttrs_t     *InAttrs,      /* IN */  
    ns_FilesetAttrBits_t  BitsForOutAttrs, /* IN */  
    ns_FilesetAttrs_t     *OutAttrs);    /* OUT */
```

Description

Set the metadata fields of the fileset identified by *FilesetHandleP* or *FilesetId*. The specific attributes to be set are identified by *InAttrBits* and the values to which these fields are to be set are found in *InAttrs*.

Parameters

| | |
|----------------------|---|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the <i>hpss_ConnectHandle</i> that defines the connection context for this user. |
| <i>UserCreds</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestId</i> | The request identifier specified by the client. |
| <i>FilesetHandle</i> | Pointer to an object handle which uniquely identifies the fileset whose attributes are to be set. If the <i>FilesetHandle</i> is NULL the <i>FilesetId</i> will be used. See the Notes below for more details. |
| <i>FilesetId</i> | Pointer to a unique fileset identifier which specifies the fileset root object whose fileset attributes are to be set. If the <i>FilesetId</i> is NULL the <i>FilesetHandle</i> will be used. See the Notes below for more details. |
| <i>InAttrBits</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is provided in the fileset attribute structure, <i>InAttrs</i> , below. |
| <i>InAttrs</i> | Pointer to the structure containing the values, if any, that are to be put into the fileset. |

| | |
|------------------------|--|
| <i>BitsForOutAttrs</i> | A bit vector in which the appropriate bit is set (on) for each attribute value that is to be returned in <i>OutAttrs</i> . |
| <i>OutAttrs</i> | Pointer to a structure containing the attributes specified by <i>BitsForOutAttrs</i> . |

Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

Error Conditions

The *ns_SetFilesetAttr* procedure is unsuccessful and the name server data base remains unchanged if any of the following are true:

| | |
|-----------------|--|
| HPSS_EACCES | The client is not the Root user. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The name server couldn't convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EINVAL | An attempt was made to change the <i>FilesetId</i> or the <i>FilesetHandle</i> . |
| HPSS_EMDATA | An inconsistency was encountered in the name server's metadata. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from or writing to the name server data base. |
| HPSS_ENOTREADY | The name server has not completed its initialization. |

See also

ns_GetFilesetAttr

Clients

Client APIs, *insif*

Notes

If both the *FilesetHandle* and *FilesetId* are non-NULL the Name Server will insure that they both point to the same object. If they do not, an error will be returned. If both are NULL an error will be returned.

Valid bits for the *InAttrBits* are defined in the data definition section.

If a *FilesetHandle* is supplied performance will be slower because the Name Server will have to access the indicated object block.

2.1.28. ns_Statistics

Purpose

Initialize and/or return the Name Server's statistics.

Syntax

```
#include <cns_interface.h>
```

```
signed32
```

```
ns_Statistics (
```

```
    trpc_handle_t          BindH,          /* IN */
    hpss_connect_handle_t *ConnectH,     /* IN */
    hsec_UserCred_t       *UserCreds,     /* IN */
    unsigned32            RequestId,      /* IN */
    unsigned32            Options,        /* IN */
    ns_StatisticsRec_t    *StatsRecord);  /* OUT */
```

Description

This procedure is used to get the Name Server's statistics or to re-initialize the Name Server's Fileset Cache. If the caller is asking for the statistics it is possible to re-initialize these statistics. See the Notes below for more information about *Options*

Parameters

| | |
|---------------------|---|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectHP</i> | Pointer to the hpss_ConnectHandle that defines the connection context for this user. |
| <i>UserCredsP</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestId</i> | The request identifier specified by the client. |
| <i>Options</i> | Used to control the Name Server's behavior while gathering the statistics. See the Notes below. |
| <i>StatsRecordP</i> | Ponter to the StatisticsRec containing the Name Server's statistics. |

Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

Error Conditions

The ns_SetAttr procedure is unsuccessful and the name server data base remains unchanged if any of the following are true:

| | |
|----------------|--|
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The name server couldn't convert the HPSS credentials into a form suitable for internal use. |

| | |
|-----------------|--|
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_ENOTREADY | The name server has not completed its initialization. |

See Also

None

Clients

SSM, insif

Notes

The statistics record returned by this call is the Name Server's global statistics record. This record contains statistics about the use of Name Server functions, errors, block usage, and other miscellaneous information. These statistics are not "long term" statistics. This is the data accumulated since the Name Server was last started or since the statistics record was last initialized.

The statistics record can be re-initialized (set to zero) by turning on the appropriate bit in the *Options* parameter. This bit can be set by or'ing in the constant "NS_REINIT_STATISTICS". When the statistics record is re-initialized, the *StatisticsStartTime* is set to the current time. The following constants can be used to set the *Options* parameter:

| | |
|---------------------------------|---|
| NS_STATS_DONT_REINIT_STATISTICS | the statistics record is returned and left unaltered. |
| NS_STATS_REINIT_STATS | the statistics record is returned, it is then set to zero, and the <i>StatisticsStartTime</i> is set to the current time. |
| NS_STATS_REINIT_FS_CACHE | the Name Server's fileset cache will be re-initialized. This is done by freeing all of the currently held heap space and then calling <i>InitFilesetCache</i> . No statistics are returned. |

2.1.29. ns_UpdateACL

Purpose

Update the ACL entries for the specified Name Server object.

Syntax

```
#include "cns_interface.h"
```

```
signed32
```

```
ns_UpdateACL (  
    trpc_handle_t          BindH,          /* IN */  
    hpss_connect_handle_t *ConnectH,     /* IN */  
    hsec_UserCred_t       *UserCred,     /* IN */  
    unsigned32            RequestID,     /* IN */  
    ns_ObjHandle_t       *Directory,     /* IN */  
    uchar                 *PathName,     /* IN */  
    unsigned32            DontBackUp,    /* IN */  
    unsigned32            Options,       /* IN */  
    ns_ACLConfArray_t    *ACLEntries);  /* IN */  
    ns_RemainingPath_t    *RemainingPath); /* OUT */
```

Description

Update the specified entries of the ACL associated with the object identified by *PathName*. If no *PathName* is supplied, the ACLs for the object identified by *Directory* are updated.

Parameters

| | |
|-------------------|--|
| <i>BindH</i> | The remote procedure call explicit binding handle. |
| <i>ConnectH</i> | Pointer to the handle that defines the connection context for this user. |
| <i>UserCred</i> | Pointer to the end user's credentials on whose behalf the request is being made. |
| <i>RequestID</i> | The request identifier specified by the client. |
| <i>Directory</i> | Pointer to the directory containing the object whose ACL is to be updated. |
| <i>PathName</i> | Pointer to the path name of the object whose ACL is to be updated. |
| <i>DontBackUp</i> | This parameter contains a boolean value. If the value is 1 and any ".." components in the <i>PathName</i> would result in backing up past the starting directory, the name server will stop processing the request and return an error. If the value is 0 then the name server will allow ".." path components to back up past the starting directory. If an error is returned, the <i>RemainingPath</i> will contain the remainder of the <i>PathName</i> . |
| <i>Options</i> | A bit vector containing bits which control the behavior of ns_UpdateACL while calculating the MASK_OBJ. See the Notes below for more information. |
| <i>ACLEntries</i> | Pointer to the ACL entries that are to be updated. |

RemainingPath Pointer to the structure containing returned information that is necessary to resolve the remainder of *PathName*.

Return values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error conditions.

Error conditions

The **ns_UpdateACL** procedure is unsuccessful and the Name Server data base remains unchanged if any of the following are true:

| | |
|-------------------|--|
| HPSS_EACCES | Search permission is denied on a component of the directory path, or the owner does not have write permission, or others do not have control permission. |
| HPSS_EAGAIN | Resources are temporarily unavailable. |
| HPSS_EBACKOVER | A <i>PathName</i> containing “..” components would have backed over the <i>PathName</i> origin and the <i>DontBackUp</i> option was set to ‘true’. |
| HPSS_EBADCONN | The supplied connection context is not formatted as expected. |
| HPSS_EBADCREDS | The Name Server couldn’t convert the HPSS credentials into a form suitable for internal use. |
| HPSS_EENTERCONN | An error was encountered while attempting to enter the connection. |
| HPSS_EIO | An internal error occurred while reading from or writing to the Name Server data base. |
| HPSS_EMDATA | An inconsistency was encountered in the name server’s metadata. |
| HPSS_EMOREPATH | The name server could not fully resolve the input <i>PathName</i> . Information to help resolve the <i>PathName</i> can be found in <i>RemainingPath</i> . |
| HPSS_ENAMETOOLONG | The length of the path name exceeds HPSS_MAX_PATH_NAME or the length of a component in the path name exceeds HPSS_MAX_FILE_NAME. |
| HPSS_ENOENT | A component of the path name does not exist. |
| HPSS_ENOTDIR | The object handle is not to a valid directory. |
| HPSS_ENOTREADY | The Name Server has not completed its initialization. |
| HPSS_ESTALE | The generation number in the <i>Directory</i> is incorrect. |

See also

ns_DeleteACL, **nsGetACL**, **ns_SetACL**.

Clients

Client APIs, insif.

Notes

Options can be used to mimic the behavior of the following `acl_edit` options: `-n`, `-c`, and `-p`. This mimicking is done using the following constants:

DONT_CALCULATE_MASK

Specifies that a new mask should not be calculated. This option is useful only for objects that are required to recalculate a new mask after they are modified. If a modify operation creates a mask that unintentionally adds permissions to an existing ACL entry, the modify causing the mask recalculation will abort with an error unless you specify either the `CALCULATE_MASK_IGNORE_ERRORS` or `DONT_CALCULATE_MASK` option.

CALCULATE_MASK_IGNORE_ERRORS

Creates or modifies the object's `MASK_OBJ` type entry with permissions equal to the union of all entries other than type `USER_OBJ`, `OTHER_OBJ`, and unauthenticated. This creation or modification is done after all other modifications to the ACL are performed. The new mask is set even if it grants permissions previously masked out. It is recommended that you use this option only if not specifying it results in an error. This option is useful only for objects that support the `MASK_OBJ` type and are required to recalculate a new mask after they are modified.

If a modify operation creates a mask that unintentionally adds permissions to an existing ACL entry, the modify causing the mask recalculation will abort with an error unless you specify either the `CALCULATE_MASK_IGNORE_ERRORS` or `DONT_CALCULATE_MASK` option.

PURGE_MASK_PERMS

Purges all masked permissions (before any other modifications are made). This option is useful only for ACLs that contain an entry of type `MASK_OBJ`. Use it to prevent unintentionally granting permissions to an existing entry when a new mask is calculated as a result of adding or modifying an ACL entry.

If an update operation creates a `MASK_OBJ` that unintentionally adds permissions to an existing ACL entry, the modify causing the `MASK_OBJ` recalculation will abort with an error unless you specify either the `CALCULATE_MASK_IGNORE_ERRORS` or `DONT_CALCULATE_MASK` option.

2.2. Data Definitions

This section describes key internal data definitions and all externally used data definitions which are provided by this subsystem. A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

Data definitions used that are defined by other subsystems are:

- `gss_token_t` in security subsystem
- `hpsoid_t` in infrastructure subsystem
- `hpss_object_handle_t` in DCE subsystem
- `hpss_server_attr_t` in storage system management

2.2.1. Access Control List Conformant Array - `ns_ACLConfArray_t`

Description

The `ns_ACLConfArray_t` is a structure that describes an ACL conformant array. Conformant arrays are used to transport ACL entries to and from the Name Server. The number of entries to be moved is placed in the Length field of the conformant array. The caller must allocate the space needed to hold the ACL entries that are put into the conformant array.

Format

The `ns_ACLConfArray_t` structure has the following format:

```
typedef struct {
    signed32          Length;
    [size_is(Length)] ns_ACLEntry_t  ACLEntry[*];
} ns_ACLConfArray_t;
```

Length

Identifies the number of ACL entries to be moved.

ACLEntry

Identifies an ACL entry.

Clients

`chacl`, Client APIs, `insif`, `loadhpssfs`, `lsacl`, `nsclientlib`

2.2.2. Access Control List Entry - `nsACLEntry_t`

Description

The `nsACLEntry_t` is a 12 byte structure that describes an ACL entry. Each entry contains information such as the type of entry (i.e., for a group or individual user), the identity and location of the user or group and the permissions that are allowed.

Format

The `nsACLEntry_t` structure has the following format:

```
typedef struct {
    unsigned char      EntryType;
    unsigned char      Perms;
```

```
    unsigned16      ExpirationDate;  
    unsigned32      EntryId;  
    unsigned32      Location;  
} ns_ACLEntry_t;
```

EntryType

Identifies the type of the EntryId field. These correspond to the POSIX ACL tag types.

Perms

Specifies the permissions or access rights.

ExpirationDate

Currently not used.

EntryId

Specifies an identifier (usually a UID or GID).

Location

Identifies the DCE cell identifier to be associated with EntryId. A value of zero on input specifies the local DCE cell.

Clients

chacl, Client APIs, dumphpssfs, insif, lsacl, nsclientlib, nsde.

2.2.3. Attribute Bit Map - ns_AttrBits_t

Description

This structure is a bit vector (0 origin array of bits). It is used to indicate which fields in a supplied attribute structure are to be used to update an object's metadata fields and which attributes are to be returned.

Format

The ns_AttrBits structure has the following format:

```
typedef u_signed64    ns_AttrBits_t;
```

ns_AttrBits fields:

```
ATTRINDEX_ACCOUNT  
ATTRINDEX_ACL_OPTIONS  
ATTRINDEX_BIT_FILE_ID  
ATTRINDEX_CLASS_OF_SERVICE  
ATTRINDEX_COMMENT  
ATTRINDEX_COMPOSITE_PERMS  
ATTRINDEX_DM_HANDLE  
ATTRINDEX_DM_HANDLE_LENGTH  
ATTRINDEX_ENTRY_COUNT  
ATTRINDEX_FAMILY_ID  
ATTRINDEX_FILESET_HANDLE  
ATTRINDEX_FILESET_ID  
ATTRINDEX_FILESET_ROOT_RSN  
ATTRINDEX_FILESET_STATE_FLAGS  
ATTRINDEX_FILESET_TYPE  
ATTRINDEX_FILE_SIZE  
ATTRINDEX_FLAGS  
ATTRINDEX_GATEWAY_UUID  
ATTRINDEX_GID
```

```

ATTRINDEX_GROUP_PERMS
ATTRINDEX_LINK_COUNT
ATTRINDEX_LOCATION
ATTRINDEX_MAC_SEC_LABEL
ATTRINDEX_OTHER_PERMS
ATTRINDEX_SET_GID_ON_EXE
ATTRINDEX_SET_STICKY_BIT
ATTRINDEX_SET_UID_ON_EXE
ATTRINDEX_TIME_LAST_READ
ATTRINDEX_TIME_LAST_WRITTEN
ATTRINDEX_TIME_OF_METADATA_UPDATE
ATTRINDEX_TYPE
ATTRINDEX_UID
ATTRINDEX_USER_PERMS

```

```
MAX_ATTR_INDEX
```

Number of attributes defined - 1.

Clients

Client APIs, insif, loadhpssdmid, loadhpsssf2.2.3.1. Name Server Attribute Structure - ns_Attrs_t

Description

The ns_Attrs_t is a structure containing fields for the various attributes (metadata) that the Name Server maintains for an object.

Format

The Name Server attributes structure has the following format:

```

typedef struct {
    unsigned32      Account;
    unsigned32      ACLOptions;
    hpssioid_t      BitFileId;
    unsigned32      ClassOfService;
    unsigned char   Comment[NS_MAX_COMMENT_LENGTH];
    unsigned32      CompositePerms;
    byte            DMHandle[MAX_DMEPI_HANDLE_SIZE];
    unsigned32      DMHandleLength;
    unsigned32      EntryCount;
    unsigned32      FamilyId;
    ns_ObjHandle_t  FilesetHandle;
    u_signed64      FilesetId;
    unsigned32      FilesetRootRSN;
    unsigned32      FilesetStateFlags;
    unsigned32      FilesetType;
    u_signed64      FileSize;
    unsigned32      Flags;
    uuid_t          GatewayUUID;
    unsigned32      GID;
    unsigned32      GroupPerms;
    unsigned32      LinkCount;
    unsigned32      Location;
    unsigned32      MACSecLabel;
    unsigned32      OtherPerms;
    unsigned32      SetGIDOnExe;
    unsigned32      SetStickyBit;
    unsigned32      SetUIDOnExe;
    timestamp_sec_t TimeLastRead;
    timestamp_sec_t TimeLastWritten;
    timestamp_sec_t TimeOfMetadataUpdate;
    unsigned32      Type;
    unsigned32      UID;
    unsigned32      UserPerms;
} ns_Attrs_t;

```

Account

Specifies opaque accounting information.

ACLOptions

Specifies the Access Control List options used with the ns_UpdateACL API.

BitField

Specifies the Bit file identifier.

ClassOfService

Specifies the class of service of a file object.

Comment

Specifies the uninterpreted client supplied ASCII text.

CompositePerms

Specifies the permission to an object after all ACLs have been examined and applied.

DMHandle

A handle that “points” back to a DMAP managed object. This field is opaque data to the Name Server.

DMHandleLength

The byte length of the DMHandle.

EntryCount

A read-only field which contains the number of entries contained in a directory. If the object is not a directory, the value is not defined.

FamilyId

Identifies the fileset FamilyId when used in operations involving filesets.

FilesetHandle

A Name Server object handle used to point to the root node of a fileset.

FilesetId

The Fileset Id that uniquely identifies the fileset an object belongs to

FilesetRootRSN

A read-only field which contains the Relative Sequence Number of the root node of this fileset.

FilesetStateFlags

This field contains flag bits indicating the state of the fileset. The following constants define the possible states:

| | |
|-----------------------|--|
| NS_FS_STATE_READ | If this bit is on reading is permitted. |
| NS_FS_STATE_WRITE | If this bit is on writing is permitted. |
| NS_FS_STATE_DESTROYED | If this bit is on the fileset has been destroyed. Neither reading nor writing will be permitted. |

NS_FS_STATE_READ_WRITE This is a combination of READ and WRITE.

NS_FS_STATE_COMBINED This is all of the above bits combined.

FilesetType

The type of the fileset the attributes are for. This is a read-only field. The following constants define the fileset types:

NS_FS_TYPE_HPSS_ONLY This fileset is a native HPSS only fileset.

NS_FS_TYPE_ARCHIVED This fileset is a backup copy of some other fileset.

NS_FS_TYPE_DFS_ONLY This fileset is native to some other file system such as DFS.

NS_FS_TYPE_MIRRORED This fileset is a mirrored copy of some other fileset such as a DFS fileset.

FileSize

Specifies the byte size of a file object.

Flags

A bit vector which contains information that can be expressed in boolean form. The following constants define the bits in this field:

NS_ATTRS_FLAGS_EXTENDED_ACLS

This bit is set to 1 if the object has extended ACL entries. Extended ACL entries are all entries other than user obj, group obj, and other obj.

GatewayUUID

The UUID of the DMAP Gateway that manages this object

GID

Specifies the principal group identifier.

GroupPerms

Specifies the permissions granted to group members.

LinkCount

Specifies the number of hard links to a file object.

Location

Specifies the DCE cell identifier.

MACSecLabel

Specifies the Mandatory Access Control Security Label.

OtherPerms

Specifies the permissions granted to 'other' clients.

SetGIDOnExe

For file objects:

0 = do not set GID to owner.

1 = set GID to owner.

SetStickyBit

For file objects:

0 = do not set the sticky bit.

1 = set the sticky bit.

SetUIDonExe

For file objects:

0 = do not set UID to owner.

1 = set UID to owner.

TimeLastRead

Specifies the last time the object was accessed.

TimeLastWritten

Specifies the last time the object was updated.

TimeOfMetadataUpdate

Specifies the last time the metadata was updated.

Type

Specifies the 'type' of the object: file, directory, junction, symbolic link, or hard link.

UID

Specifies the User Identifier of the object's owner.

UserPerms

Specifies the permissions granted to the owner of the object.

Clients

chacl, Client APIs, dumphpssfs, insif, loadhpssfs, nsclientlib

2.2.4. Name Server Directory Entry - ns_DirEntry_t

Description

The ns_DirEntry_t is a structure containing information about a directory object, such as its name, object handle and sequence index within the directory (cookie).

Format

```
typedef struct DirEntryTag{
    ns_ObjHandle_t ObjHandle;
    unsigned char    Name[HPSS_MAX_FILE_NAME];
    unsigned32       ObjOffset;
    struct DirEntryTag *Next;
    ns_Attrs_t       Attrs;
```

```
} ns_DirEntry_t;
```

HPSS_MAX_FILE_NAME is defined as 256.

ObjHandle

Specifies the object handle associated with the directory entry.

Name

Specifies the name of the directory entry.

ObjOffset

Specifies the starting location in the directory (continue cookie) from which to return a list of directory entries..

Next

Points to the next directory entry.

Attrs

Specifies the attributes of the directory entry.

Clients

Client APIs, dumphpssfs, insif, nsclientlib

2.2.5. Name Server Fileset Bit Map - ns_FilesetAttrBits_t

Description

This section describes a bit vector (0 origin array of bits) which is used to indicate which fields in the supplied ns_FilesetAttrs_t structure are to be used to update an object's metadata fields and/or which fields are to be returned.

Format

The FilesetAttrBits structure has the following format:

```
typedef unsigned ns_FilesetAttrBits_t;
```

FilesetAttrBits fields:

```
NS_FS_ATTRINDEX_REGISTER_BIT_MAP
NS_FS_ATTRINDEX_CLASS_OF_SERVICE
NS_FS_ATTRINDEX_FAMILY_ID
NS_FS_ATTRINDEX_FILESET_HANDLE
NS_FS_ATTRINDEX_FILESET_ID
NS_FS_ATTRINDEX_FILESET_NAME
NS_FS_ATTRINDEX_FILESET_TYPE
NS_FS_ATTRINDEX_GATEWAY_UUID
NS_FS_ATTRINDEX_STATE_FLAGS
NS_FS_ATTRINDEX_SUB_SYSTEM_ID
NS_FS_ATTRINDEX_USER_DATA
NS_FS_ATTRINDEX_DIRECTORY_COUNT
NS_FS_ATTRINDEX_FILE_COUNT
NS_FS_ATTRINDEX_HARD_LINK_COUNT
NS_FS_ATTRINDEX_JUNCTION_COUNT
NS_FS_ATTRINDEX_SYM_LINK_COUNT
```

```
NS_FS_MIN_ATTR_INDEX
NS_FS_MAX_ATTR_INDEX
```

```
Zero.
Number of attributes defined - 1
```

Clients

archivedel, archivelist, archiverec, Client APIs, crtjunction, dumphpssfs, insif, loadhpssfs, lsfilesets, lsvol, nsclientlib, SSM

2.2.6. Name Server Fileset Attrs structure - ns_FilesetAttrs_t

Description

The ns_FilesetAttrs_t structure is used when creating, updating, or querying a fileset.

Format

```
typedef struct
{
    u_signed64          RegisterBitMap;
    unsigned32         ClassOfService;
    unsigned32         FamilyId;
    ns_ObjHandle_t     FilesetHandle;
    u_signed64         FilesetId;
    unsigned char      FilesetName[NS_FS_MAX_FS_NAME_LENGTH];
    unsigned32         FilesetType;
    uuid_t             GatewayUUID;
    unsigned32         StateFlags;
    unsigned32         SubSystemId;
    byte               UserData[NS_FS_MAX_USER_DATA];
    signed32           DirectoryCount;
    signed32           FileCount;
    signed32           HardLinkCount;
    signed32           JunctionCount;
    signed32           SymLinkCount;
} ns_FilesetAttrs_t;
```

NS_FS_MAX_FS_NAME_LENGTH is defined as 128.

NS_FS_MAX_USER_DATA is defined as 128.

RegisterBitMap

A managed object with each bit in the bit vector corresponding to a field in the record.

ClassOfService

The Class Of Service (COS) of this fileset. See the Bitfile Server for appropriate COS values.

FamilyId

The fileset family identifier. This Id is opaque to the Name Server.

FilesetHandle

A Name Server object handle which points to the root node of the fileset.

FilesetId

The fileset identifier. This Id is opaque data to the Name Server.

FilesetName

A unique client chosen name for the fileset. Typically this is a readable ASCII string.

FilesetType

The type of the fileset. The current fileset types are HPSS_Only, Archived, DFS-Only, and Mirrored. Each of these types is DFS specific. These types can be specified with the following Name Server constants:

NS_FS_TYPE_HPSS_ONLY
 NS_FS_TYPE_ARCHIVED
 NS_FS_TYPE_DFS_ONLY
 NS_FS_TYPE_MIRRORED

GatewayUUID

The UUID of the DMAP Gateway which manages this fileset.

StateFlags

A bit vector whose bits indicate the State of the fileset. The current fileset states are:

NS_FS_STATE_READ
 NS_FS_STATE_WRITE
 NS_FS_STATE_DESTROYED
 NS_FS_STATE_READ_WRITE logical OR of READ and WRITE
 NS_FS_STATE_COMBINED logical OR of READ, WRITE, and DESTROYED

SubSystemId

This field is not currently used.

UserData

This field is not currently used.

DirectoryCount

Uninterpreted data supplied by the client. This data can be ASCII, binary, or both.

FileCount

A count of the number of files in this fileset.

HardLinkCount

A count of the number of hard links in this fileset.

JunctionCount

A count of the number of junction in this fileset.

SymLinkCount

A count of the number of symbolic links in this fileset.

Clients

archivedel, archivelist, archiverec, Client APIs, crtjunction, dumphpssfs, insif, loadhpssfs, lsfilesets, lsvol, nsclientlib, SSM

2.2.7. Name Server FilesetAttrs Conformant Array - ns_FilesetAttrsConfArray_t**Description**

The ns_FilesetAttrsConfArray_t is a structure that describes a conformant array of fileset entries. Fileset conformant arrays are used to transport fileset entries from the Name Server to a client. The number of entries being moved is placed in the Length field of the conformant array

Format

The `ns_FilesetAttrsConfArray_t` structure has the following format:

```
typedef struct
{
    signed32                Length;
    [size_is(Length)] ns_FilesetAttrsEntry_t  FilesetAttrsEntry[*];
} ns_FilesetAttrsConfArray_t;
```

Length

The number of fileset entries to be moved.

FilesetAttrsEntry

A fileset attrs entry.

Clients

Client APIs, SSM, insif, nsclientlib

2.2.8. Name Server FilesetAttrs Entry - ns_FilesetAttrsEntry_t

Description

The `ns_FilesetAttrsEntry_t` is a structure that describes a FilesetAttrs entry. Each entry contains information such as the name of the fileset, the handle to the fileset root directory, the FilesetId, the type of the fileset, the state of the fileset, and the Offset continuation cookie.

Format

The `ns_FilesetAttrsEntry_t` structure has the following format:

```
typedef struct
{
    uchar                Name[NS_FS_MAX_FS_NAME_LENGTH];
    ns_ObjHandle_t      FilesetHandle;
    u_signed64          FilesetId;
    unsigned32          FilesetType;
    unsigned32          StateFlags;
    u_signed64          Offset;
} ns_FilesetAttrsEntry_t;
```

`NS_FS_MAX_FS_NAME_LENGTH` is defined as 128.

Name

The client chosen name for the fileset.

FilesetHandle

The Name Server object handle to the fileset root directory.

FilesetId

The fileset identifier.

FilesetType

The type of the fileset. The current fileset types are HPSS-Only, Archived, DFS-Only, and Mirrored. Each of these types is DFS specific. These types can be specified with the following Name Server constants:

```
NS_FS_TYPE_HPSS_ONLY
NS_FS_TYPE_ARCHIVED
```


The `ns_GlobalFilesetEntry_t` structure has the following format:

```
typedef struct
{
    u_signed64      FilesetId;
    uchar          FilesetName[NS_FS_MAX_FS_NAME_LENGTH];
    uuid_t         GatewayUUID;
    uuid_t         NameServerUUID;
    u_signed64     Offset;
} ns_GlobalFilesetEntry_t;
```

`NS_FS_MAX_FS_NAME_LENGTH` is defined as 128.

FilesetId

The fileset identifier.

FilesetName

The name of the fileset.

GatewayUUID

The UUID of the Gateway that manages this fileset. Note that this field will be empty (zero) for HPSS_ONLY filesets.

NameServerUUID

The UUID of the Name Server that manages this fileset.

Offset

This offset, sometimes called the continuation cookie, is the offset to use in subsequent calls to `ns_ReadGlobalFileset` so that subsequent entries (if any) can be read.

Clients

Client APIs, SSM, insif, nsclientlib, nsds

2.2.11. Name Server Junction Path Conformant Array - `ns_JunctionPathConfArray_t`

Description

The `ns_JunctionPathConfArray_t` is a structure that describes a conformant array of Junction path entries. Junction path conformant arrays are used to transport Junction Path entries from the Name Server to a client. The number of entries being moved is placed in the Length field of the conformant array

Format

The `ns_JunctionPathConfArray_t` structure has the following format:

```
typedef struct
{
    signed32      Length;
    [size_is(Length)] ns_JunctionPathEntry_t      JunctionPathEntry[*];
} ns_JunctionPathConfArray_t;
```

Length

The number of Junction path entries to be moved.

JunctionPathEntry

A Junction path entry.

Clients

insif, nsclientlib

2.2.12. Name Server Junction Path Entry - ns_JunctionPathEntry_t**Description**

The ns_JunctionPathEntry_t is a structure that describes a Junction path entry. Each entry contains information such as the path name to the Junction, the FilesetHandle, the Junction handle, and the Offset continuation cookie.

Format

The ns_JunctionPathEntry_t structure has the following format:

```
typedef struct
{
    ns_ObjHandle_t    FilesetHandle;
    uchar            JunctionPathName[HPSS_MAX_PATH_NAME];
    ns_ObjHandle_t    JunctionHandle;
    u_signed64        Offset;
} ns_JunctionPathEntry_t;
```

FilesetHandle

A Name Server object handle to the fileset that contains this Junction. The JunctionPathName is relative to this object handle.

JunctionPathName

The path name to the Junction relative to the FilesetHandle.

JunctionHandle

The object handle to this Junction.

Offset

This offset, sometimes called the continuation cookie, is the offset to use in subsequent calls to ns_ReadJunctionPathNames so that subsequent entries (if any) can be read.

Clients

insif, nsclientlib

2.2.13. Name Server Object Handle - ns_ObjHandle_t**Description**

The ns_ObjHandle_t is a 32 byte structure containing information that allows the name server to identify the SFS record where the metadata for the object is stored.

Format

The Name Server object handle structure has the following format:

```
typedef struct {
    unsigned32        ObjId;
    unsigned32        FileId;
    unsigned char     Flags;
    unsigned char     Pad1;
    unsigned char     Pad2;
    unsigned char     Pad3;
```

```
        unsigned16      Generation;  
        unsigned char   Type;  
        unsigned char   Version;  
        uuid_t          NameServerUUID;  
} ns_ObjHandle_t;
```

ObjId

Specifies a unique Name Server object identifier. (The Relative Sequence Number (RSN) of the SFS record containing the metadata for the object.).

FileId

If the Type field specifies a hard link this is the RSN of the SFS record containing the metadata for the original file. For all other Types this field is equal to the ObjId.

Flags

This is field a bitvector whose bits convey additional information about the object handle. The defined bit positions for the Flags field are:

```
        NS_OH_FLAG_FILESET_ROOT
```

Pad1

Reserved for future use.

Pad2

Reserved for future use.

Pad3

Reserved for future use.

Generation

Specifies a random number used to detect stale object handles.

Type

Specifies the 'type' of the object: file, directory, symbolic link, junction, or hard link.

Version

Specifies the object handle version number.

NameServerUUID

Specifies the UUID of the Name Server that issued this object handle.

Clients

archivedel, archivelist, archiverec, crtjunction, Client APIs, dumphpssfs, insif, loadhpssdmid, loadhpssfs, lsvol, nsclientlib, nsde, recover_segs, SSM

2.2.14. Name Server Return Structure - ns_RemainingPath_t

Description

Upon return from many of the Name Server APIs the ns_RemainingPath_t structure may contain information about where future path resolution may begin. If while parsing a PathName a junction point is encountered, the remainder of the PathName (the portion not yet processed) and the object handle to the root node of the fileset are returned in an ns_RemainingPath_t structure.

Format

The ns_RemainingPath_t structure has the following format:

```
typedef struct
{
    uchar                RemainingPath[HPSS_MAX_PATH_NAME];
    ns_ObjHandle_t       DirHandle;
} ns_RemainingPath_t;
```

RemainingPath

The remainder of the PathName (the portion not yet processed)

DirHandle

The Name Server object handle found in the junction point that points to the next Directory.

Client

Client APIs, SSM, insif, nsclientlib

2.2.15. Name Server Configuration - ns_SpecificConfig_t**Description**

The ns_SpecificConfig_t structure contains the configuration information for the Name Server.

Format

The ns_SpecificConfig_t structure has the following format:

```
typedef struct {
    u_signed64    RegisterBitmap;
    uuid_t        NSId;
    uuid_t        BFSId;
    unsigned char NSACLFileName[HPSS_MAX_DCE_NAME];
    unsigned char NSFilesetAttrsFileName[HPSS_MAX_DCE_NAME];
    unsigned char NSGlobalFilesetsFileName[HPSS_MAX_DCE_NAME];
    unsigned char NSObjFileName[HPSS_MAX_DCE_NAME];
    unsigned char NSTextFileName[HPSS_MAX_DCE_NAME];
    unsigned char RootFilesetName[NS_FS_MAX_FS_NAME_LENGTH];
    u_signed64    RootFilesetId;
    unsigned32    MaxPathComponents;
    unsigned32    MaxByteSizeOfBuffer;
    unsigned32    FileDefaultPerms;
    unsigned32    DirDefaultPerms;
    unsigned32    RootIsGod;
    unsigned32    RootsUID;
    unsigned32    WarningThreshold;
    unsigned32    CriticalThreshold;
    unsigned32    WarningThresholdExceeded;
    unsigned32    CriticalThresholdExceeded;
    u_signed64    MaxRecords;
    u_signed64    NumRecordsUsed;
    u_signed64    NumRecordsAvailable;
    signed32      NumFreeRecords;
    signed32      DirectoryCount;
    signed32      FileCount;
    signed32      FilesetCount;
    signed32      HardLinkCount;
    signed32      JunctionCount;
    signed32      SymLinkCount;
} ns_SpecificConfig_t;
```

RegisterBitmap

A bitmap having a bit location corresponding to each field in this record.

NSId

The Name Server's UUID.

BFSId

The UUID of the Bitfile Server.

NSACLFileName

The path name to the SFS file where overflow ACL entries from Name Server objects are stored.

NSFilesetAttrsFileName

The path name to the SFS file where the Name Server's fileset attrs metadata is stored.

NSGlobalFilesetsFileName

The path name to the SFS file where the global fileset metadata is stored.

NSObjFileName

The path name to the SFS file where metadata for Name Server objects are stored.

NSTextFileName

The path name to the SFS file where text overflow data from Name Server objects are stored.

RootFilesetName

The name to be assigned to the Name Server's root fileset. This is the HPSS_ONLY fileset that contains the '/' directory.

RootFilesetId

The fileset Id to be assigned to the Name Server's root fileset. This is the HPSS_ONLY fileset that contains the '/' directory.

MaxPathComponents

The maximum number of path name components that the Name Server will parse.

MaxByteSizeOfBuffer

The maximum size of the buffer (in bytes) that the Name Server will use to return a list of directory entries from the ns_ReadDir function. If this buffer becomes too large the DCE RPC marshaling routines experience stack overflows.

FileDefaultPerms

The permissions that will be given to any newly inserted files that do not have permissions specified in the attribute structure.

DirDefaultPerms

The permissions that will be given to any newly created directories that do not have permissions specified in the attribute structure.

RootIsGod

Determines if root access has omnipotent power. If the value is 1, then root does have omnipotent power, otherwise, root is treated as any other user.

RootsUID

The root user's UID.

WarningThreshold

A percentage value of used space which, when exceeded in the SFS object file, will cause a notification to be sent to the SSM.

CriticalThreshold

A percentage value of used space which, when exceeded in the SFS object file, will cause alarm messages to be sent to the SSM.

WarningThresholdExceeded

A Boolean flag that is set to 1 when the warning threshold is exceeded.

CriticalThresholdExceeded

A Boolean flag that is set to 1 when the critical threshold is exceeded.

MaxRecords

The total number of SFS records that can be used by the Name Server to store object metadata.

NumRecordsUsed

The total number of SFS records that are currently being used in the object file.

NumRecordsAvailable

The total number of SFS records that are currently available in the object file. This value includes NumFreeRecords.

NumFreeRecords

The total number of SFS records that have previously been used to store Name Server objects, but are once again available to store new objects.

DirectoryCount

The number of metadata entries in the SFS object file that are for directories.

FileCount

The number of metadata entries in the SFS object file that are for files.

FilesetCount

A count of the number of filesets managed by this Name Server.

HardLinkCount

The number of metadata entries in the SFS object file that are for hard links.

JunctionCount

The number of metadata entries in the SFS object file that are for junction points.

SymLinkCount

The number of metadata entries in the SFS object file that are for symbolic links.

Clients

Name Server initialization, Storage System Manager, insif, load_cns_config, lsvol, nsclientlib,

recover_segs

2.2.16. Name Server Statistics Structure - ns_StatisticsRec_t

Description

The Name Server maintains a global data structure which contains a variety of statistics detailing Name Server activity. This activity covers the number of times each API is called and the number of times these calls resulted in an error. In addition there is information about block activity and other miscellaneous activities. This data is not preserved across Name Server restarts and can be set to zero using the ns_Statistics API.

Format

The ns_StatisticsRec_t structure has the following format:

```
typedef struct
{
    unsigned32      StatisticsStartTime;

    unsigned32      Deletes;
    unsigned32      DeleteErrors;
    unsigned32      DeleteACLs;
    unsigned32      DeleteACLErrors;
    unsigned32      DeleteFilesets;
    unsigned32      DeleteFilesetError;
    unsigned32      GetACLs;
    unsigned32      GetACLErrors;
    unsigned32      GetAttrs;
    unsigned32      GetAttrErrors;
    unsigned32      GetFilesetAttrs;
    unsigned32      GetFilesetAttrErrors;
    unsigned32      GetFilesetNameOrId;
    unsigned32      GetFilesetNameOrIdErrors;
    unsigned32      GetNames;
    unsigned32      GetNameErrors;
    unsigned32      Inserts;
    unsigned32      InsertErrors;
    unsigned32      MkDirs;
    unsigned32      MkDirErrors;
    unsigned32      MkFilesets;
    unsigned32      MkFilesetError;
    unsigned32      MkJunctions;
    unsigned32      MkJunctionErrors;
    unsigned32      MkLinks;
    unsigned32      MkLinkErrors;
    unsigned32      MkSymLinks;
    unsigned32      MkSymLinkErrors;
    unsigned32      NSGetAttrs;
    unsigned32      NSGetAttrErrors;
    unsigned32      NSSetAttrs;
    unsigned32      NSSetAttrErrors;
    unsigned32      ReadDirs;
    unsigned32      ReadDirErrors;
    unsigned32      ReadFilesetAttrs;
```

```
unsigned32    ReadFilesetAttrErrors;
unsigned32    ReadGlobalFileset;
unsigned32    ReadGlobalFilesetErrors;
unsigned32    ReadJunctionPathNames;
unsigned32    ReadJunctonPathNameErrors;
unsigned32    ReadLinks;
unsigned32    ReadLinkErrors;
unsigned32    Renames;
unsigned32    RenameErrors;
unsigned32    ServerGetAttrs;
unsigned32    ServerGetAttrErrors;
unsigned32    ServerSetAttrs;
unsigned32    ServerSetAttrErrors;
unsigned32    SetACLs;
unsigned32    SetACLErrors;
unsigned32    SetAttrs;
unsigned32    SetAttrErrors;
unsigned32    SetFilesetAttrs;
unsigned32    SetFilesetAttrErrors;
unsigned32    Statistics;
unsigned32    StatisticErrors;
unsigned32    UpdateACLs;
unsigned32    UpdateACLErrors;

unsigned32    AddedToFreeList;
unsigned32    CalledFillInFreeList;
unsigned32    AttemptedToFillFreeList;
unsigned32    FoundSomeFreeListEntries;
unsigned32    NewRecordWasInUse;
unsigned32    NewRecordInUseTooMuch;
unsigned32    UsedAnEntryFromTheFreeList;
unsigned32    AddedNewEntryToTheEnd;

unsigned32    AddedAnACLBlock;
unsigned32    DeletedAnACLBlock;
unsigned32    AddedAFilesetAttrsBlock;
unsigned32    DeletedAFilesetAttrsBlock;
unsigned32    AddedAGlobalFilesetBlock;
unsigned32    DeletedAGlobalFilesetBlock;
unsigned32    AddedATextBlock;
unsigned32    DeletedATextBlock;

unsigned32    NumFilesetCacheSlots;
unsigned32    FilesetCacheSlotsUsed;
unsigned32    RSNCacheSlotsUsed;
unsigned32    LongestFSCacheChain;
unsigned32    LongestRSNCacheChain;
unsigned32    FilesetAttrsEntryFoundInCache;
unsigned32    FilesetAttrsEntryWasNotInCache;
unsigned32    FilesetAttrsEntryWasNotInCache2;
unsigned32    DeleteFSCacheEntryCalled;
unsigned32    FSCreateAttemptWasAborted;
```

```
unsigned32    FSUpdateAttemptWasAborted;
unsigned32    GetFilesetCacheCalled;
unsigned32    GetFSCacheByRSNCalled;
unsigned32    ReloadedTheFilesetCache;

unsigned32    AccessedAComment;
unsigned32    AccessedAFilesetAttrRecord;
unsigned32    AccessedAGlobalFilesetRecord;
unsigned32    AddedALongName;
unsigned32    AddedAComment;

} ns_StatisticsRec_t;
```

StatisticsStartTime

The time in standard Unix seconds that this record was initialized (set to zero, and the StatisticsStartTime set to the current time).

Deletes

The number of times ns_Delete has been called since StatisticsStartTime.

DeleteErrors

The number of times ns_Delete has returned an error since StatisticsStartTime.

DeleteACLs

The number of times ns_DeleteACL has been called since StatisticsStartTime.

DeleteACLErrors

The number of times ns_DeleteACL has returned an error since StatisticsStartTime.

DeleteFilesets

The number of times ns_DeleteFileset has been called since StatisticsStartTime.

DeleteFilesetErrors

The number of times ns_DeleteFileset has returned an error since StatisticsStartTime.

GetACLs

The number of times ns_GetACLs has been called since StatisticsStartTime.

GetACLErrors

The number of times ns_GetACLs has returned an error since StatisticsStartTime.

GetAttrs

The number of times ns_GetAttrs has been called since StatisticsStartTime.

GetAttrErrors

The number of times ns_GetAttrs has returned an error since StatisticsStartTime.

GetFilesetAttrs

The number of times ns_GetFilesetAttrs has been called since StatisticsStartTime.

GetFilesetAttrErrors

The number of times ns_GetFilesetAttrs has returned an error since StatisticsStartTime.

GetFilesetByNameOrId

The number of times ns_GetFilesetByNameOrId has been called since StatisticsStartTime.

GetFilesetByNameOrIdErrors

The number of times ns_GetFilesetByNameOrId has returned an error since StatisticsStartTime.

GetNames

The number of times ns_GetNames has been called since StatisticsStartTime.

GetNameErrors

The number of times ns_GetNames has returned an error since StatisticsStartTime.

Inserts

The number of times ns_Inserts has been called since StatisticsStartTime.

InsertErrors

The number of times ns_Insert has returned an error since StatisticsStartTime.

MkDirs

The number of times ns_MkDir has been called since StatisticsStartTime.

MkDirErrors

The number of times ns_MkDir has returned an error since StatisticsStartTime.

MkFilesets

The number of times ns_MkFileset has been called since StatisticsStartTime.

MkFilesetErrors

The number of times ns_MkFileset has returned an error since StatisticsStartTime.

MkJunctions

The number of times ns_MkJunction has been called since StatisticsStartTime.

MkJunctionErrors

The number of times ns_MkJunction has returned an error since StatisticsStartTime.

MkLinks

The number of times ns_MkLink has been called since StatisticsStartTime.

MkLinkErrors

The number of times ns_MkLink has returned an error since StatisticsStartTime.

MkSymLinks

The number of times ns_MkSymLink has been called since StatisticsStartTime.

MkSymLinkErrors

The number of times ns_MkSymLink has returned an error since StatisticsStartTime.

NSGetAttrs

The number of times ns_NSGetAttr has been called since StatisticsStartTime.

NSGetAttrErrors

The number of times ns_NSGetAttr has returned an error since StatisticsStartTime.

NSSetAttrs

The number of times ns_NSSetAttr has been called since StatisticsStartTime.

NSSetAttrErrors

The number of times ns_NSSetAttr has returned an error since StatisticsStartTime.

ReadDirs

The number of times ns_ReadDir has been called since StatisticsStartTime.

ReadDirErrors

The number of times ns_ReadDir has returned an error since StatisticsStartTime.

ReadFilesetAttrs

The number of times ns_ReadFilesetAttrs has been called since StatisticsStartTime.

ReadFilesetAttrErrors

The number of times ns_ReadFilesetAttrs has returned an error since StatisticsStartTime.

ReadGlobalFilesets

The number of times ns_ReadGlobalFilesets has been called since StatisticsStartTime.

ReadGlobalFilesetErrors

The number of times ns_ReadGlobalFilesets has returned an error since StatisticsStartTime.

ReadJunctionPathNames

The number of times ns_ReadJunctionPathNames has been called since StatisticsStartTime.

ReadJunctionPathNameErrors

The number of times ns_ReadJunctionPathNames has returned an error since StatisticsStartTime.

ReadLinks

The number of times ns_ReadLink has been called since StatisticsStartTime.

ReadLinkErrors

The number of times ns_ReadLink has returned an error since StatisticsStartTime.

Renames

The number of times ns_Rename has been called since StatisticsStartTime.

RenameErrors

The number of times ns_Rename has returned an error since StatisticsStartTime.

ServerGetAttrs

The number of times ns_ServerGetAttr has been called since StatisticsStartTime.

ServerGetAttrErrors

The number of times ns_ServerGetAttr has returned an error since StatisticsStartTime.

ServerSetAttrs

The number of times ns_ServerSetAttr has been called since StatisticsStartTime.

ServerSetAttrErrors

The number of times ns_ServerSetAttr has returned an error since StatisticsStartTime.

SetACLs

The number of times ns_SetACL has been called since StatisticsStartTime.

SetACLErrors

The number of times ns_SetACL has returned an error since StatisticsStartTime.

SetAttrs

The number of times ns_SetAttrs has been called since StatisticsStartTime.

SetAttrErrors

The number of times ns_SetAttrs has returned an error since StatisticsStartTime.

SetFilesetAttrs

The number of times ns_SetFilesetAttrs has been called since StatisticsStartTime.

SetFilesetAttrErrors

The number of times ns_SetFilesetAttrs has returned an error since StatisticsStartTime.

Statistics

The number of times ns_Statistics has been called since StatisticsStartTime.

StatisticErrors

The number of times ns_Statistics has returned an error since StatisticsStartTime.

UpdateACLs

The number of times ns_UpdateACL has been called since StatisticsStartTime.

UpdateACLErrors

The number of times ns_UpdateACL has returned an error since StatisticsStartTime.

AddedToFreeList

The number of times since StatisticsStartTime that AddToFreeList has been called.

CalledFillInFreeList

The number of times since StatisticsStartTime that FillInFreeList has been called.

AttemptedToFillFreeList

The number of times since StatisticsStartTime the the number of FreeList entires was below the threshold and we attempted to replenish the list. This value is incremented in FillInFreeList.

FoundSomeFreeListEntries

The number of times since StatisticsStartTime that we actually found some FreeList entries in the database when we looked for some. This value is incremented in FillInFreeList.

NewRecordWasInUse

The number of times since StatisticsStartTime that we read in a FreeList record and discovered that it had already been claimed by another thread. This value is incremented in NSBlkAdd.

NewRecordInUseTooMuch

The number of times since StatisticsStartTime that 5 attempts have been made to get a FreeList entry and each time it was discovered that the record had already been claimed by some other thread. This value is incremented in NSBlkAdd.

UsedAnEntryFromTheFreeList

The number of times since StatisticsStartTime that an attempt to get a FreeList entry was successful. This value is incremented in NSBlkAdd.

AddedNewEntryToTheEnd

The number of times since StatisticsStartTime that we were not able to find a FreeList entry and added the new entry to the end of the database. This value is incremented in NSBlkAdd.

AddedAnACLBlock

The number of times since StatisticsStartTime that an ACL record has been added to the ACL Extensions file. This value is incremented in NSBlkAdd.

DeletedAnACLBlock

The number of times since StatisticsStartTime that an ACL record has been deleted. This value is incremented in NSBlkDelete.

AddedAFilesetAttrsBlock

The number of times since StatisticsStartTime that a FilesetAttrs record has been added. This value is incremented in NSBlkAdd.

DeletedAFilesetAttrsBlock

The number of times since StatisticsStartTime that a FilesetAttrs record has been deleted. This value is incremented in NSBlkDelete.

AddedAGlobalFilesetBlock

The number of times since StatisticsStartTime that a Global Fileset record has been added. This value is incremented in NSBlkAdd.

DeletedAGlobalFilesetBlock

The number of times since StatisticsStartTime that a Global Fileset record has been deleted. This value is incremented in NSBlkDelete.

AddedATextBlock

The number of times since StatisticsStartTime that a Text record has been added to the Text Extensions file. This value is incremented in NSBlkAdd.

DeletedATextBlock

The number of times since StatisticsStartTime that a Text record has been deleted. This value is incremented in NSBlkDelete.

NumFilesetCacheSlots

The total number of 'slots' or entries in the Name Server's in-memory fileset hash table.

FilesetCacheSlotsUsed

The number of slots in use in the Name Server's in-memory fileset hash table. These are the slots that have been hashed by fileset identifier.

RSNCacheSlotsUsed

The number of slots in use in the Name Server's in-memory fileset hash table. These are the slots that have been hashed by Relative Sequence Number.

LongestFSCacheChain

Fileset entries are 'chained' in link lists from the fileset hash table. This is the number of entries in the longest of these chains that is hashed by fileset identifier.

LongestRSNCacheChain

Fileset entries are 'chained' in link lists from the fileset hash table. This is the number of entries in the longest of these chains that is hashed by relative sequence number.

FilesetAttrsEntryFoundInCache

A fileset entry was requested and this FilesetAttrs entry was found in the Name Server's cache.

FilesetAttrsEntryWasNotInCache

A fileset entry was requested, but this FilesetAttrs entry was not found in the Name Server's cache.

FilesetAttrsEntryWasNotInCache2

A fileset entry was requested, and this FilesetAttrs entry was found in the Name Server's cache. However when later trying to find it again so that a copy could be made, the Name Server could not find it.

DeleteFSCacheEntryCalled

The number of times since StatisticsStartTime that DeleteFSCacheEntry has been called.

FSCreateAttemptWasAborted

The number of times since StatisticsStartTime that a Fileset create attempt was aborted.

FSUpdateAttemptWasAborted

The number of times since StatisticsStartTime that a Fileset update attempt was aborted.

GetFilesetCacheCalled

The number of times since StatisticsStartTime that the function GetFilesetCache was called.

GetFSCacheByRSNCalled

The number of times since StatisticsStartTime that the function GetFilesetCacheByRSN was called.

ReloadedTheFilesetCache

The number of times since StatisticsStartTime that the Fileset cache has been reloaded.

AccessedAComment

The number of times since StatisticsStartTime that a Comment Extension Text record has been accessed (read). This value is incremented in NSBikGet.

AccessedAFilesetAttrsRecord

The number of times since StatisticsStartTime that a FilesetAttrs record has been accessed (read).

AccessedAGlobalFilesesRecord

The number of times since StatisticsStartTime that a Global Fileset record has been accessed (read).

AddedALongName

The number of times since StatisticsStartTime that a Name Extension Text record has been added to the Text Extensions file. This value is incremented in NSBikAdd.

AddedAComment

The number of times since StatisticsStartTime that a Comment Extension Text record has been added to the Text Extensions file. This value is incremented in NSBikAdd.

Clients

Client APIs, SSM, insif

3. Bitfile Server Functions

This chapter specifies the Bitfile Server programming interface. Specifically, the following information is provided:

Application Programming Interfaces (APIs)

Data Definitions

3.1. API Functions

This section describes all APIs which are provided for use by another HPSS subsystem or by a client external to HPSS. The API interface specification includes the following information:

Name

Syntax

Description

Parameters

Return Values

Error Conditions

Related Information

Clients

Notes

3.1.1. bfs_BitfileGetAttrs

Purpose

Get attributes for a bitfile that is not required to be open.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_BitfileGetAttrs(  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNHPtr,         /* IN */  
    hpssoid_t             *BfldPtr,         /* IN */  
    reqid_t               ReqlId,          /* IN */  
    bf_attr_t             *BfAttribPtr,     /* OUT */  
    gss_token_t           Ta,              /* IN */  
    trpc_status_t         *RPCError);      /* OUT */
```

Description

The bitfile attributes are returned. Some attributes, such as current position, are not valid on this call.

Parameters

| | |
|---|---|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfldPtr->hpssoid_t</i> | The unique unforgeable identifier of the bitfile. |
| <i>ReqlId</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>BfAttribPtr->bf_attr_t</i> | Bitfile attributes. |
| <i>Ta</i> | Kerberos style authorization ticket. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|----------------|--|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EINVAL | Format of the bitfile id pointed to by BfldPtr is not valid. |
| HPSS_EPERM | Ticket passed from the name server is not valid. Note that any user with a valid ticket can get the attributes of a bitfile. No specific permissions are required. |
| HPSS_EBADCONN | Connection handle is invalid. |

| | |
|--------------|--|
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_ENOENT | Bitfile which is target of the request does not exist. |
| HPSS_ENOMEM | Memory allocation error occurred during processing of request. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |

See also

bfs_BitfileSetAttrs, bfs_BitfileOpenGetAttrs, bfs_BitfileOpenSetAttrs.

Clients

Client API, Storage System Manager.

Notes

None.

3.1.2. bfs_BitfileGetXAttrs

Purpose

Get extended attributes for a bitfile

Syntax

```
#include bfs_interface.h
[nontransactional]
signed32
bfs_BitfileGetXAttrs (
    trpc_handle_t                Binding,          /* IN */
    hpss_connect_handle_t*      CnhPtr,          /* IN */
    hpssoid_t*                  BfldPtr          /* IN */
    reqid_t                      Reqld           /* IN */
    unsigned32                   Flags           /* IN */
    unsigned32                   StorageLevel     /* IN */
    bf_xattrib_t*                BfAttribPtr     /* IN */
    gss_token_t                  Ta              /* IN */
    trpc_status_t*              RPCErr);        /* OUT */
```

Description

The bitfile information is queried for standard attributes and these are returned. In addition, the storage information associated with the bitfile is examined and information relating to the placement of the file in the hierarchy is returned to the client. This includes information as to whether the file is stored on tape or disk and, if the file is on tape, relative position of the file on the tape media. The caller can also ask only that optimization parameters be returned. The parameters returned in all cases reflect things such as the staging option associated with the class of service of the file being queried.

Parameters

| | |
|---|--|
| <i>Binding</i> | The Encina remote procedure call binding handle. |
| <i>CnhPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfldPtr->bitfile</i> | The operation is to take place on. |
| <i>Reqld</i> | The unique request id associated with the request. |
| <i>Flags</i> | Specifies options of the GetXAttrs call. Valid options are: BFS_GETATTRS_STATS_FOR_LEVEL 0x00000002 - get storage attributes associated with the storage level passed as a parameter in the call. BFS_GETATTRS_STATS_FOR_1ST_LEVEL 0x00000004 - get storage attributes associated with the bitfile associated with the first storage level in the hierarchy that contains bitfile data. BFS_GETATTRS_STATS_OPTIMIZE 0x00000008 - get optimization parameters only. The additional parameters returned are optimum access size and stripe width. |
| <i>StorageLevel</i> | Indicates the level in the hierarchy for which storage attributes should be returned. This only applies if the BFS_GETATTRS_STATS_FOR_LEVEL call is made. |

| | |
|-----------------------------------|--|
| <i>BfAttribPtr->returned</i> | Attributes. |
| <i>Ta</i> | The Kerberos style authorization ticket. |
| <i>RPCError->trpc_status_t</i> | The RPC exception status code for a non-transactional RPC. |

Return Values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error Conditions

| | |
|---------------|--|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_EPERM | Ticket passed from the name server is not valid. Note that any user with a valid ticket can get the attributes of a bitfile. No specific permissions are required. |
| HPSS_ENOENT | Bitfile which is target of the request does not exist. |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_EINVAL | Format of the bitfile id pointed to by BfIdPtr is not valid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |

See Also

bfs_BitfileGetAttrs

Clients

Client API

Notes

None.

3.1.3. `bfs_BitfileOpenGetAttrs`

Purpose

Get all attributes for the open bitfile.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_BitfileOpenGetAttrs (  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNHPtr,         /* IN */  
    hpss_object_handle_t  *BfhPtr,         /* IN */  
    reqid_t               ReqlId,          /* IN */  
    bf_attrib_t           *BfAttribPtr,    /* OUT */  
    trpc_status_t         *RPCError);     /* OUT */
```

Description

The bitfile attributes are returned, including position of the open bitfile.

Parameters

| | |
|---|---|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfhPtr->hpss_object_handle_t</i> | Bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile. |
| <i>ReqlId</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>BfAttribPtr->bf_attrib</i> | Bitfile attributes. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|-------------------|--|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by <i>BfhPtr</i> does not refer to a currently open bitfile. |

See also

`bfs_BitfileOpenSetAttrs`, `bfs_BitfileGetAttrs`, `bfs_BitfileSetAttrs`.

Clients

Client API, Storage System Manager.

Notes

None.

3.1.4. bfs_BitfileOpenSetAttrs

Purpose

Set the specified attributes in the bitfile descriptor of an open bitfile.

Syntax

```
#include "bfs_interface.h"

signed32
bfs_BitfileOpenSetAttrs (
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNHPtr,         /* IN */
    hpss_object_handle_t  *BfhPtr,         /* IN */
    reqid_t                Reqld,          /* IN */
    u_signed64             OwnerRecFlags,   /* IN */
    u_signed64             InSelectBitmap,  /* IN */
    u_signed64             *OutselectBitmap, /* OUT */
    bf_attr_t              *InBfAttribPtr,  /* IN */
    bf_attr_t              *OutBfAttribPtr,  /* OUT */
    trpc_status_t         *RPCError);      /* OUT */
```

Description

The attributes specified in the attribute structure that can be rewritten replace the current values.

Parameters

| | |
|---|---|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfhPtr->hpss_object_handle_t</i> | Bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile. |
| <i>Reqld</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>OwnerRecFlags</i> | Used to set reverse map fields in the owner record. |
| <i>InSelectBitmap</i> | Specifies the attribute fields that are to be changed. The bitfile attributes flags are specified under Data Definitions in the <i>bf_attr_t</i> structure. |
| <i>OutSelectBitmap</i> | Indicates the attribute fields that were changed. The bitfile attributes flags are specified under Data Definitions in the <i>bf_attr_t</i> structure. Setable attributes are as follows: |
| | CURRENT_POSITION DATA_LEN CREATE_TIME MODIFY_TIME WRITE_TIME READ_TIME |

OWNER_REC
 COS_ID
 ACCT
 REGISTER_BITMAP
 CONSISTENCY_FLAGS

In addition a PREALLOC flag can be set to indicate that data should be allocated based on the data length parameter passed in.

InBfAttribPtr ->bf_attrib_t Bitfile attribute values to be used to set the new attributes. See the *bf_attrib_t* structure under Data Definitions for a list of the attributes that can be changed.

OutBfAttribPtr ->bf_attrib_t Newly updated attribute values. See the *bf_attrib_t* structure under Data Definitions for a list of the attributes that can be changed.

RPCError->trpc_status_t RPC exception status code for a non-transactional RPC.

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|-------------------|---|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EINVAL | Attempt to set the Class of Service to a value that does not refer to a currently defined Class of Service. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_ENOSPACE | Attempt to add more owner rec entries than supported. |
| HPSS_EBUSY | A request is already active on this bitfile handle. |
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by <i>BfhPtr</i> does not refer to a currently open bitfile. |

See also

bfs_BitfileSetAttrs, **bfs_BitfileOpenGetAttrs**, **bfs_BitfileGetAttrs**.

Clients

Client API, Storage System Manager.

Notes

There is no user-defined metadata. *LinkCount* cannot be set through a **bfs_Bitfile(Open)SetAttrs** call. It can only be set by link and unlink calls. On one **bfs_Bitfile(Open)SetAttrs** call, reverse maps (*OwnerRec*) can be either added or deleted. Both cannot be accomplished on the same call.

3.1.5. bfs_BitfileSetAttrs

Purpose

Set the specified attributes in the bitfile descriptor of a bitfile that is not required to be open.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_BitfileSetAttrs (  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNHPtr,         /* IN */  
    hpssoid_t             *BfldPtr,         /* IN */  
    reqid_t               Reqld,           /* IN */  
    u_signed64            OwnerRecFlags,    /* IN */  
    u_signed64            InSelectBitmap,   /* IN */  
    u_signed64            OutSelectBitmap,  /* IN */  
    bf_attrib_t           *InBfAttribPtr,   /* IN */  
    bf_attrib_t           *OutBfAttribPtr,  /* OUT */  
    gss_token_t           Ta,              /* IN */  
    trpc_status_t        *RPCError);       /* OUT */
```

Description

The attributes specified in the attribute structure that can be rewritten replace the current values.

Parameters

| | |
|---|---|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfldPtr->hpssoid_t</i> | Unique unforgeable identifier of the bitfile. |
| <i>Reqld</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>OwnerRecFlags</i> | Used to set reverse map fields in the owner record. |
| <i>InSelectBitmap</i> | Specifies the attribute fields that are to be changed. The bitfile attributes flags are specified under Data Definitions in the <i>bf_attrib_t</i> structure. |
| <i>OutSelectBitmap</i> | Indicates the attribute fields that were changed. The bitfile attributes flags are specified under Data Definitions in the <i>bf_attrib_t</i> structure. |
| <i>InBfAttribPtr ->bf_attrib_t</i> | Bitfile attribute values to be used to set the new attributes. See the <i>bf_attrib_t</i> structure under Data Definitions for a list of the attributes that can be changed. |

OutBfAttribPtr ->bf_attrib_t Newly updated attribute values. See the *bf_attrib_t* structure under Data Definitions for a list of the attributes that can be changed. Settable attributes are as follows:

CREATE_TIME
 MODIFY_TIME
 WRITE_TIME
 READ_TIME
 OWNER_REC
 COS_ID
 REGISTER_BITMAP
 ACCT

Ta Kerberos style authorization ticket.

RPCError->trpc_status_t RPC exception status code for a non-transactional RPC.

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

HPSS_ENOTREADY Server not ready for processing requests.

HPSS_EINVAL Attempt to set the Class of Service to a value that does not refer to a currently defined Class of Service

Setting of DATA_LEN is not supported on a non-open version of SetAttrs.

Format of the bitfile id pointed to by *BfidPtr* is not valid.

HPSS_EBADCONN Connection handle is invalid.

HPSS_ESYSTEM Severe system error occurred during processing of request.

HPSS_ENOSPACE Attempt to add more owner rec entries than supported.

HPSS_ENOSUPPORT Attempt to set SECURITY attribute. Not supported in this release.

See also

bfs_BitfileOpenSetAttrs, bfs_BitfileOpenGetAttrs, bfs_BitfileGetAttrs.

Clients

Client API, Storage System Manager.

Notes

LinkCount cannot be set through a *bfs_BitfileSetAttrs* call. It can only be set by link and unlink calls. On one *bfs_BitfileSetAttrs* call, reverse maps (OwnerRec) can be either added or deleted. Both cannot be accomplished on the same call.

3.1.6. bfs_BitfileOpenSetCosByHints**Purpose**

Reselect a COS or select a better storage segment size for a file on disk.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_BitfileSetAttrs (
```

```
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNHPtr,         /* IN */
    hpssoid_t              *BfldPtr,        /* IN */
    unsigned32             ResetFlags,      /* IN */
    reqid_t                Reqld,          /* IN */
    hpss_cos_hints_t       *HintsPtr,      /* IN */
    hpss_cos_priorities_t *PriorPtr,      /* IN */
    hpss_cos_md_t          *COSPtr,        /* OUT */
    trpc_status_t          *RPCError);     /* OUT */
```

Description

The attributes specified in the attribute structure that can be rewritten replace the current values.

Parameters

| | |
|---|---|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfldPtr->hpssoid_t</i> | Unique unforgeable identifier of the bitfile. |
| <i>ResetFlags</i> | Option flags: BFS_RESET_SEGSIZE - Select only a new storage segment size. 0 - Select a new COS. |
| <i>Reqld</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>HintsPtr</i> | Pointer to COS hints. |
| <i>PriorPtr</i> | Pointer to COS priorities. |
| <i>COSPtr</i> | Pointer to structure for COS actually assigned. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|-----------------|--|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EINVAL | HintsPtr or PriorPtr is NULL. Request to select storage segment size and no file hints for min or max file size provided. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_BUSY_RETRY | Open file is currently active. |
| HPSS_ECONFLICT | Conflict in bitfile cache information??? |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |

See also

None.

Clients

Client API.

Notes

The ResetFlags field determines whether the COS is to be reselected or if a better storage segment should be selected for a file on disk. The reselection can only be performed if the file has no data in it at the time of request.

3.1.7. bfs_Clear

Purpose

Provides for the clearing of a specific piece of a bitfile. Maps are redone and associated storage is freed when possible.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_Clear (
```

```
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNHPtr,         /* IN */
    hpss_object_handle_t   *BfhPtr,        /* IN */
    unsigned32             Flags,          /* IN */
    reqid_t                Reqld,          /* IN */
    u_signed64             Offset,         /* IN */
    u_signed64             Length,         /* IN */
    trpc_status_t          *RPCError);     /* OUT */
```

Description

Provides for the clearing of a specific piece of a bitfile. Maps are redone and associated storage is freed when possible.

Parameters

| | |
|-----------------------------------|---|
| <i>Binding</i> | Remote procedure call Encina binding handle. |
| <i>CNHPtr</i> | Pointer to the connection handle that defines the connection context for this user. |
| <i>BfhPtr</i> | Pointer to the bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile. |
| <i>Flags</i> | Flag settings. No settings in this release. |
| <i>Reqld</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>Offset</i> | Bitfile offset where the clear should begin. |
| <i>Length</i> | Amount of data to clear. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|----------------|---|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |

| | |
|-------------------|--|
| HPSS_EPERM | File must be open for write to do a clear operation. |
| HPSS_EINVAL | Length paramater = 0. Must be greater than 0. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by <i>BfhPtr</i> does not refer to a currently open bitfile. |
| HPSS_ENOMEM | Memory allocation error occurred during processing of request. |
| HPSS_EBUSY | A request is already active on this bitfile handle. |

See also

None.

Clients

Client API.

Notes

This routine is needed for the client API so that it can support NFS.

3.1.8. bfs_Close

Purpose

Close a previously opened bitfile.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_Close (  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNHPtr,         /* IN */  
    hpss_object_handle_t  *BfhPtr,         /* IN */  
    reqid_t                ReqlId,         /* IN */  
    trpc_status_t          *RPCError);     /* OUT */
```

Description

Close will return resources associated with an open bitfile and invalidate the bitfile handle.

Parameters

| | |
|---|---|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. The value is NULL if the call is internal. |
| <i>BfhPtr->hpss_object_handle_t</i> | Bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile. |
| <i>ReqlId</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|-------------------|--|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by <i>BfhPtr</i> does not refer to a currently open bitfile. |
| HPSS_ENOMEM | Memory allocation error occurred during processing of request. |
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by <i>BfhPtr</i> does not refer to a currently open bitfile. |

| | |
|-------------|---|
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_ECOMM | Cannot communicate with storage servers to shut down any storage server sessions associated with this bitfile |
| HPSS_EBUSY | A request is already active on this bitfile handle. If the close request came from a client, the client will receive this error. If the close is generated by BFS in response to a connection being dropped, BFS delays until RPC finishes and the closes |

See also

bfs_Open.

Clients

Client API, Bitfile Server.

Notes

None.

3.1.9. bfs_CopyFile

Purpose

Copy one bitfile to another.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_Unlock (
```

```
    trpc_handle_t          Binding,           /* IN */
```

```
    hpss_object_handle_t *CNHPtr,          /* IN */
```

```
    hpss_object_handle_t *SrcBfhPtr,       /* IN */
```

```
    hpss_object_handle_t *DstBfhPtr,       /* IN */
```

```
    reqid_t                Reqld           /* IN */
```

```
    unsigned32             Flags          /* IN */
```

```
    signed32               DstStorageLevel /* IN */
```

```
    trpc_status_t          *RPCError);     /* IN */
```

Description

The data from the source bitfile is copied to the destination bitfile.

Parameters

Binding Remote procedure call binding handle.

CNHPtr->hpss_connect_handle_t
Handle that defines the connection context for this user.

SrcBfhPtr->hpss_object_handle_t
Bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the source bitfile.

DstBfhPtr->hpss_object_handle_t
Bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the source bitfile.

Reqld Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request.

Flags Flags settings:

BFS_NO_TRUNC_DEST = 0x00000002

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

HPSS_ENOTREADY Server not ready for processing requests.

HPSS_EBADCONN Connection handle is invalid.

HPSS_ESYSTEM Severe system error occurred during processing of request.

| | |
|-------------------|--|
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by BfhPtr does not refer to a currently open bitfile. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_EINVAL | Destination file must be open for exclusive access. Destination storage level is invalid. |
| HPSS_EPERM | Caller specified a destination storage level to copy to. Only authorized callers with write permission may do this. Destination file must be open for write. |
| HPSS_EBUSY | A request is already active on this bitfile handle. |

See also

None.

Clients

Bitfile Server.

Notes

This function is called internally by the BFS to move data in relation to a request to change the COS of a bitfile.

3.1.10. bfs_Create

Purpose

Create a bitfile and allocate space.

Syntax

```
#include "bfs_interface.h"
```

```
[transactional]
```

```
signed32
```

```
bfs_Create (  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNHPtr,         /* IN */  
    u_signed64             BfAttribFlags,   /* IN */  
    bf_attrib_t           *BfAttribPtr,    /* IN */  
    reqid_t               Reqld,          /* IN */  
    hpss_cos_hints_t      *HintsPtr,       /* IN */  
    hpss_cos_priorities_t *PriorPtr,       /* IN */  
    gss_token_t           Ta,              /* IN */  
    hpssoid_t             *BfldPtr,        /* OUT */  
    hpss_cos_md_t         *COSPtr         /* OUT */  
    gss_token_t           *BfTa);         /* OUT */
```

Description

bfs_Create will create a bitfile, allocate space for it, and save the information for the bitfile metadata.

Parameters

| | |
|---|---|
| <i>Binding</i> | Remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfAttribFlags</i> | Set attributes flags. |
| <i>BfAttribPtr->bf_attrib_t</i> | Pointer to the bitfile attributes struct, including <i>DataLen</i> , <i>COSId</i> , <i>ExpireTime</i> , <i>RevMap</i> , and <i>Acct</i> . |
| <i>Reqld</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>HintsPtr->hpss_cos_hints_t</i> | Address of COS hints parameters. |
| <i>PriorPtr->hpss_cos_priorities_t</i> | Address of COS priorities. |
| <i>Ta</i> | Authorization ticket. |
| <i>BfldPtr->hpssoid_t</i> | Unique identifier of the bitfile. |
| <i>COSPtr ->hpss_cos_md_t</i> | Address of the COS structure. |

**BfTa->gss_token_t* Address of a ticket containing the bitfile ID encrypted in the key of the Name Server.

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|---------------------|---|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_ENOSUPPORT | Attempt to set SECURITY attribute. Not supported in this release. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_EINVALCOS | Class of Service requested for bitfile does not exist. |
| HPSS_EINVALCOSHINTS | Hints structure passed for Class of Service selection is invalid. |
| HPSS_EPERM | Ticket passed from name server does not have insert permission indicated. |

See also

bfs_Unlink.

Clients

Client API, Bitfile Server.

Notes

None.

3.1.11. bfs_GetCOSStats

Purpose

Get storage statistics for a specific class of service.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_Lock (  
    trpc_handle_t      Binding,           /* IN */  
    hpss_object_handle_t *CNHPtr,       /* IN */  
    reqid_t            ReqId,           /* IN */  
    cos_t              COSId,          /* IN */  
    unsigned32         *MaxOpenBitfiles /* OUT */  
    u_signed64         *BlockSize       /* OUT */  
    u_signed64         *TotalBytes      /* OUT */  
    u_signed64         *FreeBytes       /* OUT */  
    trpc_status_t     *RPCError);      /* OUT */
```

Description

Returns statistics for a given COS.

Parameters

| | |
|---|---|
| <i>Binding</i> | Remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>ReqId</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>COSId</i> | ID of the Class of Service that information is being requested on. |
| <i>MaxOpenBitfiles</i> | Pointer to an area where the Bitfile Server will return the maximum number of bitfiles that may be open concurrently by the Bitfile Server. |
| <i>BlockSize</i> | Pointer to an area where the blocksize of the storage class at the top of the hierarchy for this COS is returned. |
| <i>TotalBytes</i> | Pointer to an area where the total bytes of storage present in this storage class is returned. |
| <i>FreeBytes</i> | Pointer to an area where the total number of bytes available for allocation in this COS is returned. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|----------------|---|
| HPSS_ENOTREADY | Server not ready for processing requests. |
|----------------|---|

| | |
|----------------|--|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_EINVALCOS | Class of Service requested does not exist. |

See also

None.

Clients

Client API.

Notes

1. This function is primarily intended for the NFS server. The NFS Server is the only one using this function currently.

3.1.12. bfs_Migrate

Purpose

Copy data in from one level in the hierarchy to 1 or more lower levels in the hierarchy.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_Migrate (  
    trpc_handle_t           Binding,           /* IN */  
    hpss_connect_handle_t  *CNHPtr,         /* IN */  
    hpss_object_handle_t   *BfhPtr,         /* IN */  
    reqid_t                ReqlId,          /* IN */  
    unsigned32             SrcLevel,        /* IN */  
    unsigned32             DestHierId,     /* IN */  
    unsigned32             DestLevel,     /* IN */  
    unsigned32             Flags,          /* IN */  
    u_signed64             *TotalBytesMoved, /* OUT */  
    hpss_segment_list_t   *SegListPtr     /* IN */  
    trpc_status_t         *RPCError);      /* OUT */
```

Description

Copy data from one level in a storage hierarchy to level in the hierarchy to one or more lower levels in the hierarchy. The copy is made non-transactionally.

Parameters

| | |
|-------------------|---|
| <i>Binding</i> | Remote procedure call Encina binding handle. |
| <i>CNHPtr</i> | Pointer to the connection handle that defines the connection context for this user. |
| <i>BfhPtr</i> | Pointer to the bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile. |
| <i>ReqlId</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>SrcLevel</i> | Storage level to copy from. |
| <i>DestHierId</i> | ID of the storage hierarchy to copy to. |
| <i>DestLevel</i> | Storage level to copy to. |
| <i>Flags</i> | Flag settings: BFS_MIGRATE_ALL BFS_MIGRATE_FORCE BFS_MIGRATE_PURGE_DATA |

| | |
|-----------------------------------|--|
| | BFS_MIGRATE_NO_MARK_PURGE |
| | BFS_MIGRATE_MULTIPLE_COPY_OP |
| <i>TotalBytesMoved</i> | Pointer to the number of bytes copied. |
| <i>SegListPtr</i> | Pointer to the list of storage segments that are to be copied. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|--------------------|--|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_EBUSY | A request is already active on this bitfile handle. |
| HPSS_ENOSPACE | Unable to allocate sufficient space in destination storage class. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |
| HPSS_EPERM | File must be open for write to do migration. Only authorized callers with write permission may issue migrate calls. |
| HPSS_ECONN | Communication with one or more servers needed to carry out this request failed. |
| HPSS_ENOTALLCOPIED | An error resulted in not all data being copied during this request. |

See also

bfs_Move, **bfs_Purge**.

Clients

Client API.

Notes

None.

3.1.13. bfs_Open

Purpose

Ready a bitfile for subsequent operations.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_Open (
```

```
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNHPtr,         /* IN */
    hpssoid_t              *BfldPtr,        /* IN */
    gss_token_t            Ta,              /* IN */
    unsigned32             OpenFlags,       /* IN */
    reqid_t                Reqld,          /* IN */
    hpss_object_handle_t  *BfhPtr,         /* OUT */
    trpc_status_t          *RPCError);      /* OUT */
```

Description

bfs_Open will access the bitfile metadata, verify authorization, set up the bitfile handle so the bitfile is quickly available for subsequent requests, and allocate resources that will be needed.

Parameters

| | |
|---|---|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfldPtr->hpssoid_t</i> | Unique unforgeable identifier of the bitfile. |
| <i>Ta</i> | Kerberos style authorization ticket. |
| <i>OpenFlags</i> | Operations that may be performed on the bitfile and is the logical sum of the desired flags. For example read, write, or destroy. |
| | BFS_OPEN_READ |
| | BFS_OPEN_WRITE |
| | BFS_OPEN_MODIFY |
| | BFS_OPEN_APPEND |
| | BFS_OPEN_TRUNCATE |
| | BFS_OPEN_EXCLUSIVE |
| | BFS_OPEN_NO_STAGE |
| | BFS_OPEN_MIGRATE |
| <i>Reqld</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for |

example to link log messages from various subsystems to the original client request.

BfhPtr->hpss_object_handle_t Bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile.

RPCError->trpc_status_t RPC exception status code for a non-transactional RPC.

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|-----------------|--|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_ENOSPACE | Unable to allocate sufficient space in target storage class when automatic stage on open is set. |
| HPSS_ECONN | Communication with one or more servers needed to carry out this request failed. |
| HPSS_EPERM | An attempt has been made to open a file with options that the user is not authorized to use. |
| HPSS_EINVAL | The <i>BfIdPtr</i> parameter is NULL |
| HPSS_ENOENT | Bitfile which is target of the request does not exist. |
| HPSS_EMAXBFOPEN | Too many bitfile already open in BFS. |
| HPSS_ECONFLICT | Open options specified conflict with what is allowed for the Class of Service the file belongs to. |
| | Open is for exclusive access and file is already open by another user. |
| | Open options include truncation but write is not also specified. |
| | Open options include both truncation and append. |

See also

bfs_Close.

Clients

Client API, Bitfile Server.

Notes

None.

3.1.14. **bfs_Purge****Purpose**

Reclaim space occupied by duplicated portions of a bitfile by purging bitfile data at a specified level in the storage hierarchy.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_Purge (
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNHPtr,        /* IN */
    hpss_object_handle_t  *BfhPtr,        /* IN */
    reqid_t                Reqld,         /* IN */
    unsigned32             Flags,         /* IN */
    unsigned32             StorageLevel,   /* IN */
    u_signed64             PgOffset,      /* IN */
    u_signed64             PgLen,        /* IN */
    u_signed64             *BytesPurged,   /* OUT */
    hpss_segment_list_t   *SegListPtr,    /* IN */
    trpc_status_t         *RPCError);     /* OUT */
```

Description

The specified space occupied by a bitfile that has been migrated to a lower level in the storage hierarchy is reclaimed and the metadata is updated. Purge is executed non-transactionally.

Parameters

| | |
|---------------------|---|
| <i>Binding</i> | Remote procedure call Encina binding handle. |
| <i>CNHPtr</i> | Pointer to the connection handle that defines the connection context for this user. |
| <i>BfhPtr</i> | Pointer to the bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile. |
| <i>Reqld</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>Flags</i> | Flag settings: PURGE_ALL PURGE_FORCE |
| <i>StorageLevel</i> | Level in the storage hierarchy that is to be purged. |
| <i>PgOffset</i> | Page offset. It is not used in delivery 2 and should be set to 0. |
| <i>PgLen</i> | Page length. It is not used in delivery 2 and should be set to 0. |

| | |
|-----------------------------------|---|
| <i>BytesPurged</i> | Pointer to a variable where the total number of bytes purged on this call will be returned. |
| <i>SegListPtr</i> | List of storage segment ids. Data that is stored in these storage segments will be purged. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|--------------------|---|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by BfhPtr does not refer to a currently open bitfile. |
| HPSS_ENOTSUPPORTED | Current releases of HPSS do not support specifying a purge offset and purge length in the paramaters. |
| HPSS_EPERM | File must be open for write to issues purge calls. |
| HPSS_EINVAL | StorageLevel paramaters is invalid. PURGE_ALL is not specified, but not list of storage segments to purge is provided. PURGE_ALL is specified and a storage segment list is passed. |
| HPSS_EBUSY | A request is already active on this bitfile handle. |

See also

None.

Clients

Client API.

MPS.

Notes

None.

3.1.15. `bfs_Read`**Purpose**

Read data from the HPSS to the client.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_Read (
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNHPtr,        /* IN */
    hpss_object_handle_t  *BfhPtr,        /* IN */
    reqid_t                ReqlId,        /* IN */
    unsigned32             Flags,         /* IN */
    IOD_t                  *IODPtr,       /* IN */
    IOR_t                  *IORPtr,       /* OUT */
    trpc_status_t          *RPCError);    /* OUT */
```

Description

Data is moved from the HPSS to the client as defined by the input/output descriptor. This is accomplished in conjunction with the Storage Server and the Mover.

Parameters

| | |
|---|---|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfhPtr->hpss_object_handle_t</i> | Bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile. |
| <i>ReqlId</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>Flags</i> | Series of bits to represent various processing options. BFS_READ_SEQUENTIAL |
| <i>IODPtr->iod_t</i> | Input-output descriptor that defines the entire data transfer. It has the necessary control for serial or parallel data transfer. |
| <i>IORPtr->ior_t</i> | Input-output response that defines the results of the data transfer. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|----------------|---|
| HPSS_ENOTREADY | Server not ready for processing requests. |
|----------------|---|

| | |
|-------------------|--|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by <i>BfhPtr</i> does not refer to a currently open bitfile. |
| HPSS_EINVAL | IOD does not indicate READ. IOD is invalid or specifies unsupported options. |
| HPSS_EPERM | File not open for read. |
| HPSS_EBUSY | A request is already active on this bitfile handle. |
| HPSS_ECONN | Communication with one or more servers needed to carry out this request failed. |

See also
bfs_Write.

Clients
Client API.

Notes
The IOD and IOR duplicate some of the parameters and return values of this function. For example the request ID is also in the IOD and the return value is in the IOR.

3.1.16. bfs_ServerGetAttrs**Purpose**

This is the get-server-status request as used to get the status of the Bitfile Server.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_ServerGetAttrs (
    trpc_handle_t           Binding,           /* IN */
    hpss_connect_handle_t  *CNHPtr,         /* IN */
    hpss_server_attrib_t   *OutServerData,   /* OUT */
    trpc_status_t          *RPCError);       /* OUT */
```

Description

The Bitfile Server status is obtained.

Parameters

Binding Encina remote procedure call binding handle.

CNHPtr->hpss_connect_handle_t Handle that defines the connection context for this user.

OutServerData->hpss_server_attrib_t Current attribute values.

RPCError->trpc_status_t RPC exception status code for a non-transactional RPC.

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

HPSS_ENOTREADY Server not ready for processing requests.

HPSS_EBADCONN Connection handle is invalid.

See also

bfs_ServerSetAttrs.

Clients

Storage System Manager.

Notes

None.

3.1.17. bfs_ServerSetAttrs

Purpose

This is the set-server-status request as used to set the status of the Bitfile Server.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_ServerSetAttrs (  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNHPtr,         /* IN */  
    u_signed64             InSelectBitmap,   /* IN */  
    u_signed64             OutSelectBitmap,  /* OUT */  
    hpss_server_attr_t    InServerData,    /* IN */  
    hpss_server_attr_t    OutServerData,   /* OUT */  
    trpc_status_t         *RPCError);      /* OUT */
```

Description

The Bitfile Server status is modified by the parameters set in the Bitfile Server attributes structure.

Parameters

| | |
|---|--|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>InSelectBitmap</i> | Specifies the attribute fields that are to be changed. The HPSS server attributes flags are defined in the hpss_server_attr.idl file. Settable attributes are REG_ADMINISTRATIVE_STATE and REG_REGISTER_BITMAP. |
| <i>OutSelectBitmap</i> | Indicates the attribute fields that were changed. The HPSS server attributes flags are defined in the hpss_server_attr.idl file. |
| <i>InServerData->hpss_server_attr_t</i> | Bitfile attribute values to be used to set the new attributes. |
| <i>OutServerData->hpss_server_attr_t</i> | Newly updated attribute values. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|--------------------|--|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ENOTSUPPORTED | Unsupported option requested. The unsupported options are REINIT and REPAIR. |

See also

`bfs_ServerGetAttrs.`

Clients

Storage System Manager.

Notes

None.

3.1.18. bfs_Stage

Purpose

Copy a specific part of a bitfile up to the highest, most responsive level in the storage hierarchy, but leave the original data unchanged.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_Stage (
```

```
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNHPtr,         /* IN */
    hpss_object_handle_t  *BfhPtr,         /* IN */
    reqid_t                Reql,           /* IN */
    unsigned32             Flags,          /* IN */
    unsigned32             StorageLevel,    /* IN */
    u_signed64             StOffset,       /* IN */
    u_signed64             StLen,         /* IN */
    trpc_status_t         *RPCError);     /* OUT */
```

Description

The Bitfile Server copies the desired data to the highest level in the storage hierarchy. The original data at the lower level in the hierarchy is left intact. Data in holes will not be staged and will not generate errors. The staged copy is made non-transactionally.

Parameters

| | |
|---------------------|---|
| <i>Binding</i> | Remote procedure call Encina binding handle. |
| <i>CNHPtr</i> | Pointer to the connection handle that defines the connection context for this user. |
| <i>BfhPtr</i> | Pointer to the bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile. |
| <i>Reql</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>Flags</i> | Flag settings: BFS_STAGE_ALL BFS_INTERNAL_STAGE BFS_ASYNC_CALL |
| <i>StorageLevel</i> | Level in the storage hierarchy. (This parameter is currently not used and should be set to 0.) |
| <i>StOffset</i> | Offset in the bitfile where the stage should begin. |
| <i>StLen</i> | Amount of data to stage. |

RPCError->trpc_status_t RPC exception status code for a non-transactional RPC.

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|--------------------|--|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by <i>BfhPtr</i> does not refer to a currently open bitfile. |
| HPSS_EINVAL | Invalid StorageLevel specified. StOffset specifies a position beyond the end of the bitfile. |
| HPSS_EBUSY | A request is already active on this bitfile handle. |
| HPSS_ENOTALLCOPIED | An error resulted in not all data being copied during this request. |

See also

bfs_Migrate, bfs_StageCallBack.

Clients

Client API, Bitfile Server.

Notes

None.

3.1.19. bfs_StageCallback**Purpose**

Copy a specific part of a bitfile up to the highest, most responsive level in the storage hierarchy, as a background request with callback.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_StageCallback (  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNHPtr,        /* IN */  
    hpss_object_handle_t  *BfhPtr,        /* IN */  
    reqid_t               Reqld,          /* IN */  
    unsigned32            Flags,          /* IN */  
    unsigned32            StorageLevel,    /* IN */  
    u_signed64            StOffset,       /* IN */  
    u_signed64            StLen,         /* IN */  
    bfs_callback_addr_t  *CallBackPtr,    /* IN */  
    gss_token_t           Ta,            /* IN */  
    trpc_status_t        *RPCError);     /* OUT */
```

Description

The Bitfile Server copies the desired data to the highest level in the storage hierarchy. The original data at the lower level in the hierarchy is left intact. Data in holes will not be staged and will not generate errors. The staged copy is made non-transactionally.

Parameters

| | |
|---------------------|---|
| <i>Binding</i> | Remote procedure call Encina binding handle. |
| <i>CNHPtr</i> | Pointer to the connection handle that defines the connection context for this user. |
| <i>BfhPtr</i> | Pointer to the bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile. |
| <i>Reqld</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>Flags</i> | Flag settings: BFS_STAGE_ALL BFS_INTERNAL_STAGE BFS_ASYNC_CALL BFS_CALLBACK_CALL |
| <i>StorageLevel</i> | Level in the storage hierarchy. (This parameter is currently not used and should be set to 0.) |

| | |
|-----------------------------------|--|
| <i>StOffset</i> | Offset in the bitfile where the stage should begin. |
| <i>StLen</i> | Amount of data to stage. |
| <i>CallBackPtr</i> | Pointer to address of where to send the response for the stage request. If NULL, no response is requested. |
| <i>Ta</i> | Kerberos style authorization ticket. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|--------------------|---|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by <i>BfhPtr</i> does not refer to a currently open bitfile. |
| HPSS_EINVAL | Invalid StorageLevel specified. <i>StOffset</i> specifies a position beyond the end of the bitfile. |
| HPSS_EBUSY | A request is already active on this bitfile handle. |
| HPSS_ENOTALLCOPIED | An error resulted in not all data being copied during this request. |
| HPSS_EPERM | Client not authorized to perform the stage request. |

See also

bfs_Migrate, bfs_Stage.

Clients

Client API, Bitfile Server.

Notes

None.

3.1.20. bfs_Unlink

Purpose

Destroy or unlink a bitfile.

Syntax

```
#include "bfs_interface.h"
```

```
[transactional]
```

```
signed32
```

```
bfs_Unlink (
```

```
    trpc_handle_t      Binding,          /* IN */
```

```
    hpss_object_handle_t *CNHPtr,       /* IN */
```

```
    hpssoid_t          *BfldPtr,        /* IN */
```

```
    gss_token_t        Ta,              /* IN */
```

```
    rev_map_t          *RevMapPtr,      /* IN */
```

```
    reqid_t             ReqId);         /* IN */
```

Description

If the link count is one, the bitfile will be destroyed. All metadata associated with the file will be removed and the space used will be reclaimed. If the link count is greater than one, the count is decremented and the metadata and space are left intact. The bitfile does not have to be open to unlink it.

Parameters

| | |
|---|---|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfldPtr->hpssoid_t</i> | Unique unforgeable identifier of the bitfile. |
| <i>Ta</i> | Kerberos style authorization ticket. |
| <i>RevMapPtr->rev_map</i> | Reverse mapping field for the bitfile that is to be destroyed. The contents of this field are not known to the Bitfile Server. |
| <i>ReqId</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used, for example, to link log messages from various subsystems to the original client request. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

| | |
|----------------|--|
| HPSS_ENOTREADY | Server not ready for processing requests. |
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |

| | |
|-------------|--|
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_EPERM | Name Server ticket does not indicate delete permissions. |
| HPSS_EINVAL | Attempt to unlink a bitfile and the link count is already 0. |
| HPSS_ENOENT | Bitfile which is target of the request does not exist. |

See also

bfs_Create.

Clients

Client API, Bitfile Server.

Notes

None.

3.1.21. bfs_Write

Purpose

Write data from the client to the HPSS.

Syntax

```
#include "bfs_interface.h"
```

```
signed32
```

```
bfs_Write (
```

```
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNHPtr,         /* IN */
    hpss_object_handle_t  *BfhPtr,         /* IN */
    reqid_t                ReqlId,         /* IN */
    unsigned32             Flags           /* IN */
    IOD_t                  *IODPtr,        /* IN */
    IOR_t                  *IORPtr,        /* OUT */
    trpc_status_t          *RPCError);     /* OUT */
```

Description

Data is moved from the client to the HPSS as defined by the input/output descriptor. This is accomplished in conjunction with the Storage Server and the Mover.

Parameters

| | |
|---|---|
| <i>Binding</i> | Encina remote procedure call binding handle. |
| <i>CNHPtr->hpss_connect_handle_t</i> | Handle that defines the connection context for this user. |
| <i>BfhPtr->hpss_object_handle_t</i> | Bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile. |
| <i>ReqlId</i> | Unique integer that identifies a particular request. It must be unique for the duration of the request. It can be used for example to link log messages from various subsystems to the original client request. |
| <i>Flags</i> | Series of bits to represent various processing options. Currently not used. |
| <i>IODPtr->IOD_t</i> | Input-output descriptor that defines the entire data transfer. It has the necessary control for serial or parallel data transfer. |
| <i>IORPtr->IOR_t</i> | Input-output response that defines the results of the data transfer. |
| <i>RPCError->trpc_status_t</i> | RPC exception status code for a non-transactional RPC. |

Return values

Zero indicates that the function was successful. A value less than zero indicates an error and is a code that defines the error.

Error conditions

HPSS_ENOTREADY Server not ready for processing requests.

| | |
|-------------------|--|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ESYSTEM | Severe system error occurred during processing of request. |
| HPSS_ENOMEM | Memory allocation error occurred during processing. |
| HPSS_EMDATA | SFS error detected in processing BFS metadata. |
| HPSS_EBADBFHANDLE | The bitfile handle pointed to by <i>BfhPtr</i> does not refer to a currently open bitfile. |
| HPSS_EINVAL | IOD does not indicate WRITE. IOD is invalid or specifies unsupported options. |
| HPSS_EPERM | File not open for write. |
| HPSS_EBUSY | A request is already active on this bitfile handle. |
| HPSS_ECONN | Communication with one or more servers needed to carry out this request failed. |
| HPSS_ENOSPACE | Unable to allocate sufficient space in storage class being written to. |

See also
bfs_Read.

Clients
Client API.

Notes
The IOD and IOR duplicate some of the parameters and return values of this function. For example the request ID is also in the IOD and the return value is in the IOR.

3.1.21.1. Data Definitions

This section describes key internal data definitions and all externally used data definitions that are provided by this subsystem. A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

3.1.22. Bitfile Volatile and Metadata Attributes - `bf_attrib_t`

Description

The attributes structure for the bitfile object contains all the volatile and metadata bitfile attributes. These are parameters relating to a bitfile.

Structure use— API parameters.

Format

The `bf_attrib` has the following format:

```
typedef struct bf_attrib {
    u_signed64    CurrentPosition;
    signed32     OpenCount;
    bf_attrib_md_t BfAttribMd;
} bf_attrib_t;
```

(BfAttribFlags— a parameter of `bfs` Create and various other functions)

Parameter is used to indicate which fields are to be modified on a set-attributes call. Each bit corresponds to a particular attribute that can be set. The bit settings are as follows:

BFS_SET_CURRENT_POSITION

BFS_SET_DATA_LEN

BFS_SET_CREATE_TIME

BFS_SET_MODIFY_TIME

BFS_SET_WRITE_TIME

BFS_SET_READ_TIME

BFS_SET_OWNER_REC

BFS_SET_COS_ID

BFS_SET_ACCT

BFS_SET_SECURITY

BFS_SET_REGISTER_BITMAP

(OwnerRecFlags— a parameter of `bfs` Create, various other functions)

Parameter is used to indicate which RevMap records in the owner record are being referenced for adding or deleting. Each bit corresponds to a reverse map field in an `owner_rec` array. If the bit is 1, this means to delete the corresponding entry from the owner record. If the bit is 0, then add the RevMap to the owner

record. This set of flags is meaningful only if SET_OWNER_REC in BfAttribFlags is also set.

CurrentPosition

The current byte position in the bitfile.

OpenCount

The current number of clients that have the bitfile open.

BfAttribMd

is the struct of bitfile metadata attributes that are stored in the data base.

Clients

The following clients access the data definition:

Client API.

3.1.23. Bitfile Metadata Attributes - bf_attr_md_t

Description

This structure contains the bitfile attributes metadata. These are parameters relating to a bitfile.

LinkCount is always 1 for a existing bitfile in current HPSS release. On one bfs_Bitfile(Open)SetAttrs call, reverse maps (OwnerRec) can be either added or deleted. Both cannot be accomplished on the same call.

Structure use - dynamic memory tables, API parameters, and permanent data base.

Format

The bf_attr_md has the following format:

```
typedef struct bf_attr_md {
    u_signed64      DataLen;
    signed32       ReadCount;
    signed32       WriteCount;
    signed32       LinkCount;
    timestamp_sec_t CreateTime;
    timestamp_sec_t ModifyTime;
    timestamp_sec_t WriteTime;
    timestamp_sec_t ReadTime;
    cos_t          COSId;
    cos_t          NewCOSId;
    acct_rec_t     Acct;
    unsigned32     Flags;
    unsigned32     StorageSegMult;
    bfs_owner_rec_t OwnerRec;
    u_signed64     RegisterBitmap;
    security_t     Security;
} bf_attr_md_t;
```

DataLen

The number of bytes of actual data that the bitfile contains.

ReadCount

The count of the number of times that all or part of the bitfile has been read.

WriteCount

The count of the number of times that data has been written to the bitfile.

LinkCount

The number of links to this bitfile. This also indicates how many reverse map IDs are in the bf_rev_map record for this bitfile.

CreateTime

The date and time the bitfile metadata was created.

ModifyTime

The date and time the bitfile metadata was last modified.

WriteTime

The date and time when data was last written to the bitfile.

ReadTime

The date and time when the bitfile was last read.

COSId

The class of service type (unsigned32) and indicates which of several classes of service the bitfile is in. This ID references a class of service record that defines the parameters for this particular class of service.

NewCOSId

Indicates the new class of service that a file is to be changed to when the client changes the class of service on a bitfile. When the change has been completed, the value of this field is moved into COSId and this field is cleared.

Acct

The accounting metadata for the bitfile. It includes information needed to charge for data storage, access, transfers, quotas, etc.

Flags

The flag settings. Not currently used.

StorageSegMult

Defines the storage segment multiple used to adjust the size of disk storage segments.

OwnerRec

Defines the reverse map entries for a bitfile and indicates which ones are active or null.

RegisterBitmap

Used to indicate the fields in the attributes structure that the SSM wants to receive notifications for when the field changes.

BFS_REG_OPEN_COUNT

BFS_REG_DATA_LEN

BFS_REG_READ_COUNT

BFS_REG_WRITE_COUNT

BFS_REG_LINK_COUNT
 BFS_REG_CREATE_TIME
 BFS_REG_MODIFY_TIME
 BFS_REG_WRITE_TIME
 BFS_REG_READ_TIME
 BFS_REG_OWNER_REC
 BFS_REG_COS_ID
 BFS_REG_ACCT
 BFS_REG_SECURITY

This vector is also set to indicate which fields in the attributes structure have changed on notify requests. If the REG_OWNER_REC field is set, then the *SetRevMapFlags* field in the bf_attr struct will be set to indicate which reverse map entries have changed.

Security

Not currently used.

Clients

The following clients access the data definition:

Client API.

3.1.24. Bitfile Descriptor - bf_descriptor_md_t

Description

This structure contains the description of the bitfile and is the header for the storage allocated and occupied by the bitfile.

Structure use - dynamic memory tables and permanent data base.

Format

The bf_descriptor_md has the following format:

```

typedef struct bf_descriptor_md {
    hpssoid_t          BfId;
    bf_attr_md_t      BfAttrib;
    bf_level_stats_md_t LevelStats[HPSS_MAX_STORAGE_LEVELS];
    u_signed64        reserv1[2];
    unsigned32        FamilyID;
    signed32          reserv2[1];
} bf_descriptor_md_t;
  
```

Bfld

The unique bitfile identifier, which is a SOID.

BfAttrib

The structure that contains all the parameters and statistics relating to a bitfile.

LevelStats[HPSS MAX STORAGE LEVELS]

The array of storage level statistics.

HPSS MAX STORAGE LEVELS

The metadata version number for this data structure. The current value is 5.

reserv1[]

Space for growth.

FamilyID

A numeric value that associates the bitfile with a particular HPSS file family.

reserv2[]

Space for growth.

Clients

The following clients access the data definition:

None.

3.1.25. Bitfile Storage Level Statistics - bf_level_stats_md

Description

This structure defines the storage level statistics, which are stored in metadata.

Structure use - dynamic memory tables and permanent data base.

Format

The bf_level_stats_md has the following format:

```
typedef struct bf_level_stats_md {
    unsigned32      Flags;
    unsigned32      Pad;
    timestamp_sec_t ReadTime;
    timestamp_sec_t WriteTime;
    timestamp_sec_t MigrateTime;
    timestamp_sec_t CacheTime;
    signed32        ReadCount;
    signed32        WriteCount;
} bf_level_stats_md_t;
```

Flags

Not currently used.

ReadTime

The date and time when the bitfile was last read.

WriteTime

The date and time when data was last written to the bitfile.

MigrateTime

The date and time when the bitfile data was last migrated to a lower level in the storage hierarchy.

CacheTime

The date and time when the bitfile was last cached for quick access.

ReadCount

The count of the number of times that all or part of the bitfile has been read.

WriteCount

The count of the number of times that data has been written to the bitfile.

Clients

The following clients access the data definition:

None.

3.1.26. Bitfile Managed Object Data Structure - bfMO_attrib_t

Description

This structure defines the bitfile managed object data structure. It is used primarily for notifications to the SSM.

Structure use - dynamic memory tables.

Format

The bfMO_attrib has the following format:

```
typedef struct bfMO_attrib {
    signed32      Version;
    hpsoid_t      BfId;
    bf_attrib_t   BfAttrib;
} bfMO_attrib_t;
```

BfId

The unique unforgeable identifier of the bitfile.

BfAttrib

Provides the attributes of the bitfile.

Clients

The following clients access the data definition:

SSM.

3.1.27. Bitfile Open Context - bf_open_context

Description

This structure is used to keep track of the dynamic control information for bitfiles that are unique to a given connection.

Structure use - dynamic memory tables.

Format

The bf_open_context has the following format:

```
typedef struct bf_open_context bf_open_context_t;
struct bf_open_context {
```

```
signed32          Index;
pthread_mutex_t  Mutex;
int              ActiveReqCount;
unsigned32       OpenFlags;
unsigned32       Flags;
hpssoid_t        BfId;
bf_cache_entry_t *BfCachePtr;
u_signed64       CurrentPosition;
signed32         SSFlags [BFS_MAX_SS_PER_OPEN];
hpssoid_t        SSSessionId [BFS_MAX_SS_PER_OPEN];
uuid_t           SSId [BFS_MAX_SS_PER_OPEN];
unsigned16       LevelAccessed;
} ;
```

Index

The index in the array of bitfile open context structures in a `bf_open_context_list`.

Mutex

Used to serialize access to this table.

ActiveReqCount

The number of requests using the bitfile.

OpenFlags

Define the requested access to the bitfile. The values are defined in the `bfs_Open` API.

Flags

Not currently used.

BfId

The bitfile ID, a SOID.

BfCachePtr

Pointer to a bitfile cache entry.

CurrentPosition

The current byte position of the file.

SSFlags

The status flags for the Storage Server. The only value currently defined indicates a valid session to the Storage Server: `SS_SESSION_VALID`.

SSSessionId

The array of all the Storage Server sessions actively operating on this bitfile.

SSId

The array of all the Storage Server IDs actively operating on this bitfile.

BFS_MAX_SS_PER_OPEN

The maximum number of Storage Servers that can be associated with an open file. The current value is 5.

LevelAccessed

Used to optimize file accesses for accounting.

Clients

The following clients access the data definition:

None.

3.1.28. Bitfile Open Context List - `bf_open_context_list_t`**Description**

This structure is a header structure used to control bitfile open context information.

Structure use - dynamic memory tables.

Format

The `bf_open_context_list` has the following format:

```
typedef struct bf_open_context_list {
    pthread_mutex_t      Mutex;
    signed32             MaxCount;
    unsigned             FreeCount;
    bf_open_context_hdr_t *ContextListPtr;
} bf_open_context_list_t;
```

Mutex

Used to serialize access to this table.

MaxCount

The maximum number of bitfiles that can be open in the Bitfile Server at once.

FreeCount

The number of open context blocks that are unused.

ContextListPtr

Points to an array of structures with each element in the array controlling the information associated with one open bitfile

Clients

The following clients access the data definition:

None.

3.1.29. Bitfile Open Context Header - `bf_open_context_hdr_t`**Description**

This structure is a description of one element of an array that is used to control the user of bitfile open context information.

Structure use - dynamic memory tables.

Format

The `bf_open_context_hdr` has the following format:

```
typedef struct bf_open_context_hdr {
    signed32             Index;
    int                 Inuse;
```

```
    bf_open_context_hdr_t *NextConnectPtr;  
    bf_open_context_t     *OpenContextPtr;  
} bf_open_context_hdr_t;
```

Index

Identifies which element in the array of headers this structure represents.

InUse

This entry and the associated `bf_open_context` is in use.

NextConnectPtr

Open context information is kept on a list off the connection handle . When a file is opened with a given connection, the bitfile open context informational is chained to this list via the **bf_open_context** header. This pointer has the address of the next one on the chain.

OpenContextPtr

Points to the open context associated with this entry. The open context structure is malloc'ed and open and freed when file is closed.

Clients

The following clients access the data definition:

none

3.1.30. Bitfile Tape Segment Metadata - `bf_tape_segment_md_t`

Description

This structure defines a bitfile's Storage Server storage location for tape media, the metadata structure for a bitfile segment on tape.

Structure use - dynamic memory tables and permanent data base.

Format

The `bf_tp_segment_md` has the following format:

```
typedef struct bf_tape_segment_md {  
    hpsoid_t      BfId;  
    unsigned32   StorageClass;  
    unsigned32   Flags;  
    hpsoid_t     SSegId;  
    u_signed64   SSegOffset;  
    u_signed64   BfSegLength;  
    u_signed64   BfOffset;  
    timestamp_t  ReadTime;  
    timestamp_t  WriteTime;  
    timestamp_t  MigrateTime;  
    timestamp_t  CacheTime;  
} bf_tape_segment_md_t;
```

BfId

The unique unforgeable identifier of the bitfile.

StorageClass

Represents the type of data storage.

Flags

Set of flags that give information about the state of a bitfile tape segment.

SSSegId

The identification of the storage segment as assigned by the Storage Server.

SSSegOffset

The byte offset in the storage segment where a data write starts.

BfSegLength

The amount of data written into this bitfile segment in bytes.

BfOffset

The byte offset (starting address) of this bitfile segment within the logical bitfile.

ReadTime

The date and time when the bitfile segment was last read.

WriteTime

The date and time when data was last written to the bitfile segment.

MigrateTime

The date and time when the bitfile segment was last migrated to a lower level in the storage hierarchy.

CacheTime

The date and time when the bitfile segment data was last cached for quick access.

Clients

The following clients access the data definition:

None.

3.1.31. Bitfile Disk Segment Metadata - `bf_disk_segment_md_t`

Description

This structure defines a bitfile's Storage Server storage location for disk media, the metadata structure for a bitfile segment on disk.

Structure use - dynamic memory tables and permanent data base.

Format

The `bf_disk_segment_md` has the following format:

```
typedef struct bf_disk_segment_md {
    hpsoid_t BfId;
    unsigned32 StorageClass;
    unsigned32 Flags;
    u_signed64 BfSegLength;
    u_signed64 BfOffset;
} bf_disk_segment_md_t;
```

BfId

The unique unforgeable identifier of the bitfile.

StorageClass

Represents the type of data storage.

Flags

Set of flags that give information about the state of a bitfile disk segment.

BfSegLength

The amount of data written into this bitfile segment in bytes.

BfOffset

The byte offset (starting address) of this bitfile segment within the bitfile where this bitfile segment starts.

Note:

Access statistics are kept in the disk map.

Clients

The following clients access the data definition:

None.

3.1.32. Bitfile Disk Segment Region - bf_disk_region_md_t

Description

This structure defines the storage allocation and statistics for a fixed length portion of a bitfile stored on disk.

Structure use - dynamic memory tables.

Format

The bf_disk_region_md has the following format:

```
typedef struct bf_disk_region_md {
    hpsoid_t      SSegId;
    timestamp_t  ReadTime;
    timestamp_t  WriteTime;
    timestamp_t  MigrateTime;
    timestamp_t  CacheTime;
    unsigned32   Flags;
    unsigned32   Pad;
} bf_disk_region_md_t;
```

SSegId

The identifier of the storage segment .

ReadTime

The date and time when the bitfile segment was last read.

WriteTime

The date and time when data was last written to the bitfile segment.

MigrateTime

The date and time when the bitfile segment was last migrated to a lower level in the storage hierarchy.

CacheTime

The date and time when the bitfile segment was last cached for quick access.

Flags

Set of status flags for a bitfile allocation map entry.

Pad

Pad for 64-bit alignment.

Clients

The following clients access the data definition:

None.

3.1.33. Bitfile Disk Allocation Map Metadata - `bf_disk_alloc_rec_md_t`**Description**

This structure defines a bitfile's allocation map metadata record.

Structure use - permanent data base.

Format

The `bf_disk_alloc_rec_md` has the following format:

```
typedef struct bf_disk_alloc_rec_md {
    hpsoid_t          BfId;
    unsigned32       RecordNumber;
    unsigned32       StorageClass;
    unsigned32       StorageSegmentSize;
    unsigned32       Pad;
    bf_disk_segment_entry_t RegionInfo[HPSS_MAX_DISK_SEGS_PER_MAPREC];
} bf_disk_alloc_rec_md_t;
```

BfId

The unique unforgeable identifier of the bitfile.

RecordNumber

The record number, starting with 0.

StorageClass

Represents the type of data storage.

StorageSegmentSize

The fixed size of storage segments for this bitfile.

Pad

Pad for 64-bit alignment.

SegmentInfo[HPSS_MAX_DISK_SEGS_PER_MAPREC]

The array of disk segment entries.

HPSS_MAX_DISK_SEGS_PER_MAPREC

The maximum number of entries in a disk map record. The current value is 8.

Clients

The following clients access the data definition:

None.

3.1.34. Class of Service - `hpss_cos_md_t`

Description

Class of service defines a set of parameters associated with operational and performance characteristics of bitfiles and associates the parameters with a hierarchy. Each bitfile in HPSS has a COS associated with it.

Structure use - dynamic memory tables and permanent data base.

Format

The COS has the following format:

```
typedef struct hpss_cos_md {
    unsigned32    COSId;
    unsigned32    HierId;
    char          COSName [HPSS_MAX_OBJECT_NAME];
    unsigned64    OptimumAccessSize;
    unsigned32    Flags;
    unsigned64    MinFileSize;
    unsigned64    MaxFileSize;
    unsigned32    AccessFrequency;
    unsigned32    TransferRate;
    unsigned32    AvgLatency;
    unsigned32    WriteOps;
    unsigned32    ReadOps;
    unsigned32    StageCode;
} hpss_cos_md_t;
```

COSId

The class of service type and indicates which of several classes of service the bitfile is in.

HierId

The identification of the hierarchy in which the bitfile is stored. This in effect points to another structure that defines the actual hierarchy. The bitfile may reside on tape, on disk, or on both.

COSName [HPSS MAX OBJECT NAME]

The name of the class of service for this bitfile.

HPSS MAX OBJECT NAME

The maximum length for a class of service name. The current value is 32.

OptimumAccessSize

The block size in bytes for this class of service that yields the maximum data transfer rate.

Flags

Used to indicate special options for the class of service. Valid values are:

| | |
|---------------------------|--|
| COS_ENFORCE_MAX_FILE_SIZE | Files in COS are not allowed to grow greater than the max file size for COS. |
|---------------------------|--|

| | |
|---------------------|---|
| COS_FORCE_SELECTION | User must ask for COS explicitly by id or name. |
|---------------------|---|

MinFileSize

The minimum size in bytes of a bitfile in this class of service.

MaxFileSize

The maximum size in bytes to which the bitfile can grow and remain in this class of service.

AccessFrequency

The expected rate of access for the bitfile.

FREQ_HOURLY

FREQ_DAILY

FREQ_WEEKLY

FREQ_MONTHLY

FREQ_ARCHIVE

TransferRateThe approximate file transfer rate in kilobytes per second.

AvgLatency

The time in seconds from when a request is received by a Storage Server until data actually begins to transmit. This is typically non-zero for tape media.

WriteOps

The valid write operations for the bitfile.

ReadOps

The valid read operations for the bitfile.

These are the read and write operation bit field flags:

OP_WRITE

OP_APPEND

OP_READ

StageCode

Represents the valid stage codes. These are the valid stage code settings:

COS_STAGE_NO_STAGE

COS_STAGE_ON_OPEN

COS_STAGE_ON_OPEN_ASYNC

COS_STAGE_ON_OPEN_BACKGROUND

Clients

The following clients access the data definition:

Client API, Storage System Management, Storage Server.

3.1.35. Class of Service Hints - `hpss_cos_hints_t`

Description

The class of service hints structure assists a client in selecting a COS for a bitfile.

Structure use - dynamic memory tables.

Format

The COS hints has the following format:

```
typedef struct hpss_cos_hints {
    unsigned32    COSId;
    char          COSName [HPSS_MAX_OBJECT_NAME];
    unsigned64    OptimumAccessSize;
    unsigned64    MinFileSize;
    unsigned64    MaxFileSize;
    unsigned32    AccessFrequency;
    unsigned32    TransferRate;
    unsigned32    AvgLatency;
    unsigned32    WriteOps;
    unsigned32    ReadOps;
    unsigned32    StageCode;
    unsigned32    StripeWidth;
    u_signed64    StripeLength;
} hpss_cos_hints_t;
```

COSId

The class of service type and indicates which of several classes of service the bitfile is in.

COSName [HPSS_MAX_OBJECT_NAME]

The name of the class of service for this bitfile.

HPSS_MAX_OBJECT_NAME = 32

The maximum length for a class of service name.

OptimumAccessSize

The block size in bytes for this class of service that yields the maximum data transfer rate.

MinFileSize

The minimum size in bytes of a bitfile in this class of service.

MaxFileSize

The maximum size in bytes to which the bitfile can grow and remain in this class of service.

AccessFrequency

The expected rate of access for the bitfile.

FREQ_HOURLY

FREQ_DAILY

FREQ_WEEKLY

FREQ_MONTHLY

FREQ_ARCHIVE

TransferRate

The approximate file transfer rate in kilobytes per second.

AvgLatency

The time in seconds from when a request is received by a Storage Server until data actually begins to transmit. This is typically non-zero for tape media.

WriteOps

The valid write operations for the bitfile.

ReadOps

The valid read operations for the bitfile.

These are the read and write operation bit field flags:

OP_WRITE

OP_APPEND

OP_READ

StageCode

Indicates which staging option should be selected.

Clients

The following clients access the data definition:

Client API, Storage System Management, Storage Server.

3.1.36. Class of Service Priorities - `hpss_cos_priorities_t`**Description**

The class of service priorities structure assists a client in selecting a COS for a bitfile.

Structure use - dynamic memory tables.

Format

The COS priorities has the following format:

```
typedef struct hpss_cos_priorities {
    unsigned32 COSIdPriority;
    unsigned32 COSNamePriority;
    unsigned32 OptimumAccessSizePriority;
    unsigned32 MinFileSizePriority;
    unsigned32 MaxFileSizePriority;
    unsigned32 AccessFrequencyPriority;
    unsigned32 TransferRatePriority;
    unsigned32 AvgLatencyPriority;
    unsigned32 WriteOpsPriority;
    unsigned32 ReadOpsPriority;
    unsigned32 StageCodePriority;
    unsigned32 StripeWidthPriority;
    unsigned32 StripeLengthPriority;
} hpss_cos_priorities_t;
```

COSIdPriority

The class of service ID priority for the class of service the bitfile should be in.

COSNamePriority

The class of service name priority for this bitfile.

OptimumAccessSizePriority

The priority for the block size for this class of service that yields the maximum data transfer rate.

MinFileSizePriority

The priority for the minimum size in bytes of a bitfile in this class of service.

MaxFileSizePriority

The priority for the maximum size in bytes to which the bitfile can grow and remain in this class of service.

AccessFrequencyPriority

The priority for the expected rate of access for the bitfile.

TransferRatePriority

The priority for the class of service file transfer rate.

AvgLatencyPriority

The class of service priority for the average latency time from request time until data begins to transfer.

WriteOpsPriority

The priority for the valid write operations for the bitfile.

ReadOpsPriority

The priority for the valid read operations for the bitfile.

StageCodePriority

The priority for the desired stage code.

StripeWidthPriority

The priority to be associated with stripe width selection. Stripewidth information comes from the underlying storage class metadata.

StripeLengthPriority

The priority to be associated with stripe length selection. Stripe length selection comes from the underlying storage class metadata..

Following are the possible priority values:

NO_PRIORITY

LOWEST_PRIORITY

LOW_PRIORITY

DESIRED_PRIORITY

HIGHLY_DESIRED_PRIORITY

REQUIRED_PRIORITY

Clients

The following clients access the data definition:

Client API, Storage System Management, Storage Server.

3.1.37. Owner Record - `bfs_owner_rec_t`

Description

This structure defines the reverse map entries for a bitfile and indicates which ones are active or null.

Format

The `bfs_owner_rec` has the following format:

```
typedef struct bfs_owner_rec {
    signed32    RevMapCount;
    unsigned32  Pad;
    rev_map_t   RevMap[BFS_NUM_REV_MAPS];
} bfs_owner_rec_t;
```

RevMapCount

The number of valid reverse map entries.

Pad

Pad for 64-bit alignment.

RevMap[BFS_NUM_REV_MAPS]

The array of opaque reverse mapping fields supplied by a client.

BFS_NUM_REV_MAPS

The total number of reverse maps in the RevMap array. The current value is 1.

Clients

The following clients access the data definition:

Client API, Storage Server.

3.1.38. Request Attributes - `req_attrib_t`

Description

This is a structure that describes the attributes or status of a Bitfile Server request.

Structure use - dynamic memory tables.

Format

The `req_attrib` has the following format:

```
typedef struct req_attrib {
    pthread_mutex_t    Mutex;
    int                WaitCount;
    char                UserName[HPSS_MAX_USER_NAME];
    char                HostName[HPSS_MAX_HOST_NAME];
}
```

```
    u_signed64          DataLen;
    struct timeval      StartTime;
    struct timeval      EndTime;
    signed32            ErrorCode;
    hpss_object_handle_t BfHandle;
    hpssoid_t           BfId;
} req_attrib_t;
```

Mutex

The mutex used to serialize access to this structure.

WaitCount

The count of requests waiting for the same resource.

UserName [HPSS_MAX_USER_NAME]

The login name of the user associated with the request.

HPSS_MAX_USER_NAME

The maximum length of a user name. The current value is 16.

HostName [HPSS_MAX_HOST_NAME]

The text name of the host that the user resides on, where the request came from.

HPSS_MAX_HOST_NAME

The maximum length of a host name. The current value is 64.

DataLen

The total amount of data in bytes to be moved by this request.

StartTime

The time at which the request started processing.

EndTime

The time at which the request completed processing.

ErrorCode

The error code associated with a request if an error occurred.

BfHandle

The bitfile handle and contains the information necessary for the Bitfile Server to find the cached metadata for the bitfile.

BfId

The ID of the bitfile for the request.

Clients

The following clients access the data definition:

None.

3.1.39. Reverse Map Field - rev_map_t

Description

Rev_map is the opaque reverse mapping field supplied by the client. The contents of this field are not known to the Bitfile Server. For example, this field can be used to map from a bitfile ID to a Name Server directory ID.

Structure use— dynamic memory tables, API parameters, and permanent data base.

Format

The rev_map has the following format:

```
typedef struct rev_map {
    byte    RevMapId [BFS_REV_MAP_LEN];
} rev_map_t;
```

RevMapId [BFS REV MAP LEN]

The reverse mapping field.

BFS REV MAP LEN

The length of the reverse mapping field. The current value is 32.

Clients

The following clients access the data definition:

Name server, Client API.

3.1.40. Bitfile Cache Entry - bf_cache_entry_t

Description

This structure is the dynamic memory data and control information needed for a bitfile that is in use. If metadata such as the bitfile descriptor are being updated, the *MetaDataLock* must be locked before the update. If other variables or pointers are being updated, the Mutex must be locked.

Structure use - dynamic memory tables.

Format

The bf_cache_entry has the following format:

```
typedef struct bf_cache_entry    bf_cache_entry_t;

struct bf_cache_entry {
    bf_cache_entry_t            *NextPtr;
    bfs_lock_cb_t              MetaDataLock;
    int                         CacheState;
    unsigned32                  Flags;
    unsigned32                  OpenCount;
    bfs_schedl_info_t           SchdlInfo;
    hpss_cos_md_t               *COSPtr;
    hpss_hier_md_t              *HierPtr;
    signed32                     StorageLevelCount;
    bf_descriptor_md_t          BfDescriptor;
    bf_segments_cache_entry_t    BfSegmentsCache[HPSS_MAX_STORAGE_LEVELS];
};
```

NextPtr

Pointer to the next bitfile cache entry in the chain from the bitfile hash table.

MetaDataLock

The lock for metadata such as the bitfile descriptor.

CacheState

The state of the cache.

Flags

The flags variable (not used).

OpenCount

The number of currently active opens for this bitfile.

SchdInfo

Structure used to schedule and control multiple requests directed at the same bitfile.

COSPtr

Pointer to the Class of Service information.

HierPtr

Pointer to the Hierarchy information

StorageLevelCount

The number of storage levels.

BfDescriptor

The copy of the bitfile descriptor.

BfSegmentsCache [HPSS MAX STORAGE LEVELS]

The array that caches information about all levels in the storage hierarchy.

HPSS MAX STORAGE LEVELS

The maximum number of storage levels. The current value is 5.

Clients

The following clients access the data definition:

None.

3.1.41. Bitfile Cache Hash - bf_cache_hash_t

Description

This structure is used to optimize searches for objects pointed at by handles. It stores hash entries to index into arrays of pointers. It also allows a safe address (not pointing to memory obtained via malloc) to be used in handles.

Structure use - dynamic memory tables.

Format

The bf_cache_hash has the following format:

```
typedef struct bf_cache_hash {
    pthread_mutex_t  Mutex;
    bf_cache_entry_t *HashPtr[BFS_MAX_BF_CACHE_HASH];
} bf_cache_hash_t;
```

Mutex

Used to serialize access to this table.

HashPtr [BFS MAX BF CACHE HASH]

An array of pointers to hash chains of bitfile cache entries.

BFS MAX BF CACHE HASH

The maximum number of entries in the cache hash. The current value is 128.

Clients

None.

3.1.42. Bitfile Segments Cache Entry - bf_segments_cache_entry_t**Description**

This structure contains cached information about the bitfile segments associated with a given level in the storage hierarchy.

Structure use - dynamic memory tables.

Format

The bf_segments_cache_entry has the following format:

```
typedef struct bf_segments_cache_entry {
    unsigned32      StorageClass;
    unsigned32      StorageLevel;
    unsigned32      SegmentCount;
    unsigned32      Flags;
    hpss_sclass_md_t *SClassPtr;
    bfs_lock_cb_t   SegmentsLock;
    void            *BfSegmentsPtr;
    bf_disk_map_t   *DiskMapPtr;
    current_segment_info_t CurSegInfo;
} bf_segments_cache_entry_t;
```

StorageClass

The class of data storage used for the bitfile segments.

StorageLevel

The level in the hierarchy represented by this cache entry.

SegmentCount

Count of the number of segments in the bitfile at this level.

Flags

Flag settings

BFS_BITFILE_SEGMENTS_LOCK

BFS_BITFILE_CHECK_CURRENT_SEG

SClassPtr

Points to the storage class information associated with data at this level.

SegmentsLock

Used to lock out access to bitfile segments.

BfSegmentsPtr

Pointer to the list of the current segments for the bitfile.

DiskMapPtr

The address of the disk map.

CurSegInfo

The current allocate segment information, used by *bfs_Write*.

Clients

The following clients access the data definition:

None.

3.1.43. Storage Segment Delete Entry - `sseg_delete_entry_t`

Description

This structure defines a storage segment delete entry. It is used in tape processing to generate a list of storage segments that have been overwritten and are candidates for deletion.

Structure use - dynamic memory tables.

Format

The `sseg_delete_entry` has the following format:

```
typedef struct sseg_delete_entry {
    struct sseg_delete_entry *NextPtr;
    hpsoid_t                  SSegmentId;
    uuid_t                    SSId;
} sseg_delete_entry_t;
```

NextPtr

Pointer to the next storage segment delete entry on the list.

SSegmentId

The storage segment ID.

SSId

The ID of the Storage Server containing the segment.

Clients

The following clients access the data definition:

None.

3.1.44. Current Bitfile Segment Information - `current_segment_info_t`

Description

This structure defines current segment information used by *bfs_Write*, for tape only.

Structure use - dynamic memory tables.

Format

The `current_segment_info` has the following format:

```
typedef struct current_segment_info {
    hpsoid_t           CurrentSSSegId;
    u_signed64        CurrentSSSegOffset;
    unsigned32        Flags;
    bf_tape_segment_cached_t *NewBfSegmentsPtr;
    sseg_delete_entry_t *SSDeleteListPtr;
    u_signed64        NextSSSegOffset;
    signed32          CurrentBlockSize;
} current_segment_info_t;
```

CurrentSSSegId

The current storage segment doing allocate from.

CurrentSSSegOffset

The next write position in the segment at the start of a write.

Flags

Control flags.

NewBfSegmentsPtr

Pointer to the list of new bitfile segments.

SSDeleteListPtr

Pointer to the list of storage segments to delete.

NextSSSegOffset

The next position to write in the segment.

CurrentBlockSize

The blocksize associated with the current segment.

Clients

The following clients access the data definition:

None.

3.1.45. Bitfile Disk Map - `bf_disk_map_t`

Description

This structure defines a record which caches the map allocated space for a bitfile stored on disk.

Structure use - dynamic memory tables.

Format

The `bf_disk_map` has the following format:

```
typedef struct bf_disk_map {
    hpsoid_t           BfId;
    unsigned32        StorageClass;
    unsigned32        StorageSegmentSize;
```

```
    unsigned32          Flags;  
    unsigned32          EntryCount;  
    bf_disk_region_md_t *DiskSegmentsPtr;  
} bf_disk_map_t;
```

Bfld

The bitfile ID.

StorageClass

The storage class.

StorageSegmentSize

The fixed storage segment size for this bitfile.

Flags

The status flags.

EntryCount

The number of entries.

DiskSegmentsPtr

The address of the disk segments list.

Clients

The following clients access the data definition:

None.

3.1.46. Bitfile Server Connect Context - `bfs_connect_context_t`

Description

This structure provides the link from the `hpss_connect_context` to the `bf_open_context` structure.

Structure use - dynamic memory tables.

Format

The `bfs_connect_context` has the following format:

```
typedef struct bfs_connect_context {  
    pthread_mutex_t      Mutex;  
    long                 Flags;  
    bf_open_context_hdr_t *BfOpenContextPtr;  
} bfs_connect_context_t;
```

Mutex

The type of data storage used for the bitfile segments.

Flags

The connect context flags.

BfOpenContextPtr

Pointer to the open context structure.

Clients

The following clients access the data definition:

None.

3.1.47. HPSS Segment List - `hpss_segment_list_t`

Description

This structure defines the storage segment list.

Structure use - parameter on `bfs_Migrate` and `bfs_Purge` calls

Format

The `hpss_segment_list` has the following format:

```
typedef struct hpss_segment_list {
    unsigned32      Flags;
    unsigned32      Count;
    hpss_segment_desc_t *SegDescPtr;
} hpss_segment_list_t;
```

Flags

The storage segment list flags.

Count

The number of storage segment descriptor entries stored in the list.

SegDescPtr

Pointer to the storage segment descriptor list.

Clients

The following clients access the data definition:

MPS.

3.1.48. HPSS Segment Descriptor - `hpss_segment_desc_t`

Description

This structure defines the storage segment descriptor.

Structure use - parameter to `bfs_Migrate`, `bfs_Purge`

Format

The `hpss_segment_desc` has the following format:

```
typedef struct hpss_segment_desc {
    struct hpss_segment_desc *NextPtr;
    unsigned32      Flags;
    hpssoid_t       SSegId;
} hpss_segment_desc_t;
```

NextPtr

Next descriptor on list

Flags

The storage segment list flags.

SSegId

ID of the storage segment.

Clients

The following clients access the data definition:

MPS.

3.1.49. HPSS Background Stage CallBack Structure - bfs_callback_addr_t

Description

This structure defines the structure containing the address information for where to send the response from a background stage request.

Format

The bfs_callback_addr has the following format:

```
typedef struct bfs_callback_addr {
    signed32    addr;
    signed32    port;
    signed32    family;
    signed32    id;
} bfs_callback_addr_t;
```

addr

Host address.

port

Port number.

family

Address family.

id

ID returned from the request.

Clients

The following clients access the data definition:

Client API.

3.2. Other Interfaces (OFD and Request list)

These interfaces are for the Open File Descriptor (OFD) manager, which uses the Encina Structured File System (SFS) to make OFDs reusable.

Also, there are interfaces for the Request List manager, which contains general purpose structures for managing the list that contains server request state. Each element in the list will contain information about a request that is currently being processed or has recently been processed by the server.

Both of these sub-systems are used internally in the Bitfile Server and in the Storage Server.

3.2.1. hpss_InitOfdMgr**Purpose**

Initialize the OFD manager.

Syntax

```
#include hpss_ofd.h
```

```
signed32  
hpss_InitOfdList ();
```

Description

Initialize the OFD header for the list of hpss_ofd_t control blocks. Initialize the mutex that serializes access to the OFD free list.

Parameters

None.

Return Values / Error Conditions

Zero indicates that the function was successful.

A value of -1 indicates an error. The system errno parameter will be set to

EINVAL out of memory for initialization,

EAGAIN out of necessary resources.

Related Information

hpss_GetOfd, hpss_FreeOfd, hpss_CloseAllOfds.

Clients

BFS, SS.

Notes

None.

3.2.2. `hpss_GetOfd`

Purpose

Acquire an OFD for use by the caller.

Syntax

```
#include hpss_ofd.h
signed32
hpss_GetOfd (
    hpss_ofd_t    **OfdPtr,    /* OUT */
    char*         FileNamePtr, /* IN */
    unsigned32    Flags        /* IN */
);
```

Description

Acquire or allocate an OFD to be used by the caller for Encina SFS calls. The OFD pointer is returned by reference in the function argument list.

In searching for a reusable OFD on the free-list, we have to find a match with *entry_ptr.SfsFileName* and the correct *Flags* attributes.. It must be opened for the file we are accessing. Check the free chain first to see if an already open OFD is available. If not, then allocate one, send an RPC to Encina SFS to open it, and return it. Different threads can reuse an OFD if the *SfsFileName* and the *Flags* match and if the former transaction is complete. These OFD reuse conditions are all handled automatically.

Parameters

| | |
|--------------------|--|
| <i>OfdPtr</i> | is a pointer reference to the OFD pointer returned. |
| <i>FileNamePtr</i> | is the name of the OFD file. |
| <i>Flags</i> | indicate what type of OFD to allocate and various attributes of the OFD. The flags can be used to select a transactional or non-transactional ofd and to set any attribute that is supported for ofds. |

Return Values / Error Conditions

The pointer to the acquired OFD is returned by the reference *OfdPtr* argument.

Zero indicates that the function was successful.

| | |
|------------------|----------------------------|
| OFD_SUCCESS | successful, |
| OFD_BAD_FILENAME | bad filename |
| OFD_MALLOC_ERROR | malloc() error |
| HPSS_EMUTEX | failure of mutex operation |
| OFD_OPEN_ERROR | OFD SFS-file open error |

Related Information

`hpss_InitOfdMgr`, `hpss_FreeOfd`, `hpss_CloseAllOfds`.

Clients

HPSS servers including BFS and SS.

Notes

3.2.3. `hpss_FreeOfd`

Purpose

Place an already open OFD on the free list (reuse list), or dispose of a bad OFD.

Syntax

```
#include hpss_ofd.h

signed32
hpss_FreeOfd (
    hpss_ofd_t    *OfdPtr,    /* IN */
    signed32      OfdStatus); /* IN */
```

Description

Free an OFD for reuse by placing the OFD on the free list. If the *OfdStatus* argument indicates a bad OFD, then dispose of the OFD.

Parameters

| | |
|------------------|---|
| <i>OfdPtr</i> | Address of the OFD returned to the free list. |
| <i>OfdStatus</i> | Marks a bad OFD for disposal. <code>OFD_BAD = -1</code> . |

Return values / Error conditions

Zero indicates that the function was successful.

| | |
|--------------------------|----------------------------|
| <code>OFD_FAILURE</code> | failure of OFD operation |
| <code>HPSS_EMUTEX</code> | failure of mutex operation |

See also

`hpss_InitOfdList`, `hpss_GetOfd`, `hpss_CloseAllOfds`.

Clients

Bitfile Server, Storage Server.

Notes

None.

3.2.4. hpss_CloseAllOfds**Purpose**

Close all open OFDs on the free list; close all OFDs as they return from use.

Syntax

```
#include hpss_ofd.h

signed32
hpss_CloseAllOfds ();
```

Description

Close all open OFDs found on the free list. Set a flag that causes all OFDs returned to `hpss_FreeOfd()` to be closed as well.

Parameters

None.

Return values / Error conditions

Zero (OFD_SUCCESS) indicates that the function was successful.

HPSS_EMUTEX = -2005; failure of mutex operation

See also

hpss_CleanupOfds

Clients

Bitfile Server, Storage Server.

Notes

This closes only ofds that are in the ofd pool. Any ofd that has not been put back into the pool is the responsibility of the application.

3.2.5. hpss_CleanupOfds

Purpose

Close ofds left open by prior invocation of server.

Syntax

```
#include hpss_ofd.h
signed32
hpss_CleanupOfds(
char          **server_files
);
```

Description

Called by a server at startup time to cleanup any ofds left open by a prior invocation of the server on the files that are passed int the paramater list. The server should be labeling ofds to use this.

Parameters

server_files-> list of character strings that provide the names of the SFS files that should be processed.

Return Values / Error Conditions

Zero (OFD_SUCCESS) indicates that the function was successful.

HPSS_EMUTEX failure of mutex operation

Related Information

hpss_CloseAllOfds

Clients

HPSS server including BFS and SS.

Notes

3.2.7. hpss_ReqListDeleteEntry

Purpose

Remove a request list entry.

Syntax

```
#include hpss_req_list.h

signed32
hpss_ReqListDeleteEntry (
    hpss_reqlist_t      *ListPtr,      /* IN */
    hpss_reqlist_entry_t *EntryPtr);  /* IN */
```

Description

Delete an entry from the request list. Lock the list. Scan the list to find the node previous to the *EntryPtr*. Unlink the entry to be deleted. Decrement the list entries count. Unlock the list. Free the storage for *EntryPtr*.

Parameters

| | |
|-----------------|--|
| <i>ListPtr</i> | Pointer to the request list from which the entry is to be deleted. |
| <i>EntryPtr</i> | Pointer to the entry to be deleted. |

Return values / Error conditions

Zero (HPSS_E_NOERROR) indicates that the function was successful.

HPSS_REQ_NULL_ENTRY Entry argument contained a NULL pointer.

Error variable Mutex lock or unlock errors. The system *errno* parameter will be set to EINVAL or EAGAIN.

HPSS_REQ_ENTRY_NOT_FOUND
The argument *EntryPtr* was not found in the request-list indicated by *ListPtr*.

See also

hpss_ReqListInit, hpss_ReqListInsertEntry, hpss_ReqListFindReqId, hpss_ReqListNextEntry, hpss_ReqListSetState.

Clients

Bitfile Server, Storage Server.

Notes

None.

3.2.8. `hpss_ReqListFindReqId`

Purpose

Find the entry that corresponds to the `ReqId`.

Syntax

```
#include hpss_req_list.h

hpss_reqlist_entry_t *
hpss_ReqListFindReqId (
    hpss_reqlist_t *ListPtr,      /* IN */
    signed32 ReqId);             /* IN */
```

Description

Find an entry on the request list that corresponds to a request identifier. This routine can be used to monitor state for a particular request.

Parameters

| | |
|----------------|---|
| <i>ListPtr</i> | Pointer to the request list in which to find the <code>ReqId</code> . |
| <i>ReqId</i> | ID number for the request. |

Return values / Error conditions

A NULL return value means the list was traversed and `ReqId` was not found; or, there was a mutex lock/unlock error.

Otherwise, a pointer to the entry with the desired `ReqId` is returned.

See also

`hpss_ReqListInit`, `hpss_ReqListInsertEntry`, `hpss_ReqListDeleteEntry`,
`hpss_ReqListNextEntry`, `hpss_ReqListSetState`.

Clients

Bitfile Server, Storage Server.

Notes

None.

3.2.10. hpss_ReqListInsertEntry**Purpose**

Add a new entry to the request list.

Syntax

```
#include hpss_req_list.h

hpss_reqlist_entry_t *
hpss_ReqListInsertEntry (
    hpss_reqlist_t *ListPtr,           /* IN */
    void *Context,                    /* IN */
    signed32 InitialState,             /* IN */
    signed32 ReqCode,                  /* IN */
    signed32 ReqId,                    /* IN */
    void *ServerSpecific,              /* IN */
    pthread_t ThreadId);               /* IN */
```

Description

Insert a new request entry at the front of the list and return a pointer to the entry. Use **malloc** to create a new request list entry node. Initialize its contents from function argument parameters. Lock the list; link the new node in at the front of the list; increment the node count; unlock the list.

Parameters

| | |
|-----------------------|---|
| <i>ListPtr</i> | Pointer to the request list. |
| <i>Context</i> | Pointer to the context of the request. |
| <i>InitialState</i> | Used to set the initial state field of the request. |
| <i>ReqCode</i> | Name or type of request. |
| <i>ReqId</i> | ID for the request. |
| <i>ServerSpecific</i> | Pointer to the server-specific state. |
| <i>ThreadId</i> | ID of the thread doing the processing. |

Return values / Error conditions

| | |
|--------------|--|
| NULL | Indicates that the new request-list node entry could not be inserted. |
| Other values | Indicates that the new entry was inserted; a pointer to the new entry is returned. |

See also

hpss_ReqListInit, **hpss_ReqListDeleteEntry**, **hpss_ReqListFindReqId**,
hpss_ReqListNextEntry, **hpss_ReqListSetState**.

Clients

Bitfile Server, Storage Server.

Notes

None.

3.2.11. `hpss_ReqListNextEntry`

Purpose

Get the next entry from the request list.

Syntax

```
#include hpss_req_list.h

hpss_reqlist_entry_t *
hpss_ReqListNextEntry (
    hpss_reqlist_t      *ListPtr      /* IN */
    hpss_reqlist_entry_t *EntryPtr); /* IN */
```

Description

Find the next entry on the request list following the argument *EntryPtr*. If the argument *EntryPtr* is NULL, then return the head of the list.

Parameters

| | |
|-----------------|---|
| <i>ListPtr</i> | Pointer to the request list to initialize. |
| <i>EntryPtr</i> | Pointer to the entry previously returned, the list entry immediately preceding the entry returned by the routine. |

Return values / Error conditions

A NULL return value means that *EntryPtr* was the last node on the request list. Otherwise, a pointer to the next entry on the request list is returned.

See also

hpss_ReqListInit, *hpss_ReqListInsertEntry*, *hpss_ReqListDeleteEntry*,
hpss_ReqListFindReqId, *hpss_ReqListSetState*.

Clients

Bitfile Server, Storage Server.

Notes

This routine does NOT check the validity of the *ListPtr* or *EntryPtr* arguments.

3.2.12. hpss_ReqListSetState**Purpose**

Set the state for the request *EntryPtr*.

Syntax

```
#include hpss_req_list.h
```

```
signed32
```

```
hpss_ReqListSetState(
    hpss_reqlist_t      *ListPtr          /* IN */
    hpss_reqlist_entry_t *EntryPtr,      /* IN */
    signed32            ReqState);       /* IN */
```

Description

Set the state for the request *EntryPtr* to the value of the argument *ReqState*.

Parameters

| | |
|-----------------|--|
| <i>ListPtr</i> | Pointer to the request list. |
| <i>EntryPtr</i> | Pointer to the entry for which state is to be set. |
| <i>ReqState</i> | The new state for request. |

Return values / Error conditions

| | |
|---------------------------|--|
| Zero (HPSS_E_NOERROR = 0) | Indicates that the function was successful. |
| HPSS_REQ_NULL_ENTRY | EntryPtr argument is NULL. |
| HPSS_REQ_ENTRY_NOT_FOUND | The EntryPtr argument was not found in the request-list. |
| HPSS_EMUTEX | Mutex error locking or unlocking list. |
| HPSS_REQ_INVALID_STATE | ReqState argument is out of range. |

See also

hpss_ReqListInit, hpss_ReqListInsertEntry, hpss_ReqListDeleteEntry, hpss_ReqListFindReqId, hpss_ReqListNextEntry.

Clients

Bitfile Server, Storage Server.

Notes

None.

3.3. Other Data Definitions (OFD and request list)

This section describes the data definitions used by the Open File Descriptor (OFD) manager and the Request List manager.

A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

3.3.1. **HPSS Open File Descriptor (OFD) - `hpss_ofd_t`**

Description

`hpss_ofd` is the control block used to get a handle on an open metadata file (Encina SFS file). This structure constitutes the node entry in the OFD free list, the list of OFDs that are available for reuse.

Structure use— dynamic memory tables.

Format

The `hpss_ofd` has the following format:

```
typedef struct hpss_ofd {
    char          OfdVerify[OFD_STR_LEN];
    struct hpss_ofd *NextPtr;
    char          SfsFileName[CELL_NAME_LEN];
    signed32     Status;
    pthread_t     ThreadId;
    sfs_ofd_t     Ofd;
    signed32     TranType;
} hpss_ofd_t;
```

OfdVerify[OFD_STR_LEN]

The data-integrity-check string field. It is set to "hpss_ofd" when an `hpss_ofd` node is created.

OFD_STR_LEN

The length of the data-integrity-check string field. The current value is 16.

NextPtr

Pointer to the next entry on the list of free OFDs.

SfsFileName[CELL_NAME_LEN]

The SFS file name associated with the OFD.

CELL_NAME_LEN

The (DCE) maximum length of the SFS file name associated with an OFD. The current value is 1024.

Status

The flag to indicate current OFD status:

HPSS_OFD_FREE

HPSS_OFD_IN_USE

ThreadId

The ID of the thread using the OFD.

Ofd

The SFS OFD (open file descriptor), similar to a UNIX file descriptor).

TranType

Indicates whether the OFD is transactional or non-transactional.

HPSS_OFD_TRANSACTIONAL

HPSS_OFD_NON_TRANSACTIONAL

Clients

The following clients access the data definition:

Bitfile server, Storage server.

3.3.2. HPSS Open File Descriptor List Header - hpss_ofd_hdr_t**Description**

The OFD header structure, hpss_ofd_t, designates a free list (Encina SFS OFD reuse list) of hpss_ofd_t node entries.

Structure use— dynamic memory tables.

Format

The hpss_ofd_hdr has the following format:

```
typedef struct hpss_ofd_hdr {
    pthread_mutex_t    Mutex;
    hpss_ofd_t         *HeadPtr;
    hpss_ofd_t         *TailPtr;
    unsigned            TotalCount;
    unsigned            InUseCount;
    unsigned            FreeCount;
} hpss_ofd_hdr_t;
```

Mutex

The DCE mutex serial access protection for the free list of OFDs.

HeadPtr

Pointer to the first entry on the free list.

TailPtr

Pointer to the last entry on the free list.

TotalCount

The total number of OFDs, both allocated and free, to be re-used.

InUseCount

The number of OFDs in use or allocated.

FreeCount

The number of OFDs available on the free list for re-use.

Clients

The following clients access the data definition:

Bitfile server, Storage server.

3.3.3. Request List - `hpss_reqlist_t`

Description

This general purpose structure is for managing the list that will contain server request state. Each element in the list will contain information about a request that is currently being processed or has recently been processed by the server. This structure is the head of the request list queue. It keeps track of the entries in the list and has access protection for the list.

Structure use - dynamic memory tables.

Format

The `hpss_reqlist` has the following format:

```
typedef struct hpss_reqlist {
    pthread_mutex_t      Mutex;
    hpss_reqlist_entry_t *HeadPtr;
    unsigned32           Count;
} hpss_reqlist_t;
```

Mutex

The mutex used to serialize access to the request list.

HeadPtr

Pointer to the first request entry on the list.

Count

The count of the number of entries on the request list.

Clients

The following clients access the data definition:

The request list is used internally by servers.

3.3.4. Request List Entry - `hpss_reqlist_entry_t`

Description

This is the entry that is queued if a request is being processed or is waiting. It contains the current state for a request. The *ReqId* and *Context* fields may be used to identify the request. The *State* field contains a value that describes the current state of the request. The *ServerSpecific* field points to server specific information.

Structure use - dynamic memory tables.

Format

The `hpss_reqlist_entry` has the following format:

```
typedef struct hpss_reqlist_entry hpss_reqlist_entry_t;

struct hpss_reqlist_entry {
    hpss_reqlist_entry_t *NextPtr;
    void                  *Context;
```

```
signed32          ReqCode;  
signed32          ReqId;  
signed32          State;  
void              *ServerSpecific;  
pthread_t         ThreadId;  
};
```

NextPtr

Pointer to the next request list entry in the list.

Context

Pointer to the context of the request; the DCE context handle. The Context could be used for a ConnectHandlePtr.

ReqCode

The code that defines the name or type of the request.

```
BFS_CREATE_REQ  
BFS_OPEN_REQ  
BFS_CLOSE_REQ  
BFS_UNLINK_REQ  
BFS_READ_REQ  
BFS_WRITE_REQ  
BFS_SET_ATTRIB_BF_REQ  
BFS_SET_ATTRIB_BF_O_REQ  
BFS_GET_ATTRIB_BF_REQ  
BFS_GET_ATTRIB_BF_O_REQ  
BFS_SET_ATTRIB_BFS_REQ  
BFS_GET_ATTRIB_BFS_REQ
```

ReqId

is the unique integer that identifies a particular request.

State

is the generic request state.

ServerSpecific

is the server-specific information.

ThreadId

is the identification of the thread doing the processing.

Clients

The following clients access the data definition:

This structure is used internally by servers.

4. Storage Server Functions

This chapter specifies the Storage Server programming interface. Specifically, the following information is provided:

Application Programming Interfaces (APIs)

Data Definitions

4.1. API Functions

This section describes all APIs which are provided for use by another HPSS subsystem or by a client external to HPSS. The API interface specification includes the following information:

Name

Syntax

Description

Parameters

Return Values

Error Conditions

Related Information

Clients

Notes

4.1.1. **ss_BeginSession**

Purpose

Start a Storage Server session.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_BeginSession (  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNH,             /* IN */  
    hpss_object_handle_t  *Session,         /* OUT */  
    signed32              ReqstID,          /* IN */  
    trpc_status_t         *RPCStatus);      /* OUT */
```

Description

The **ss_BeginSession** function starts a Storage Server session for the caller. A session is used to group a set of Storage Server requests with the resources they use. The resources used by the requests share state and compete with resources owned by other sessions on a session by session basis. The purpose of sessions is to provide a mechanism to optimize media mounts and unmounts, and to recover the use of resources allocated to the session.

Parameters

| | |
|------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The session handle returned by the function that is to be used in subsequent functions to identify the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|-------------------------------|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_EFAULT | DCE utility failure. |
| HPSS_ENOMEM | Server memory exhausted. |

See also

ss_EndSession.

Clients

BitFile Server.

Storage System Manager

Notes

This is a non-transactional function.

4.1.2. **ss_EndSession**

Purpose

End a Storage Server session.

Syntax

```
#include "ss_interface.h"

signed32
ss_EndSession (
    rpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,           /* IN */
    hpss_object_handle_t *Session,        /* IN */
    unsigned32            Options,         /* IN */
    signed32              ReqstID)         /* IN */
    trpc_status_t        *RPCStatus);     /* OUT */
```

Description

ss_EndSession terminates a Storage Server session. Resources associated with the session are released and made available to competing sessions. Storage maps reserved by the session are returned to free state, allowing space to be allocated from the virtual volumes. Mounted volumes are unmounted or transferred to other sessions waiting to use them.

Parameters

| | |
|------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Options</i> | Option flags that control details of the termination of the session. The SS_FORCE_TAPE_DISMOUNT flag in the <i>Options</i> field forces the tape storage server to dismount all tapes owned by the session immediately. Without this option, tapes are dismounted after a period of time during which they may be transferred to other sessions requesting them. |
| <i>Session</i> | The session handle that identifies the session to be terminated. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|-----------------------------------|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_EINVAL | <i>Session</i> handle is invalid. |

See also

ss_BeginSession.

Clients

Bitfile Server, Storage System Manager.

Notes

This is a non-transactional function.

This function does not block. If the session is in use when this function is called, the function will return immediately without error. The session will no longer be valid if used in any subsequent server functions, but will remain active in the server until the function using the session completes. The session will then be ended.

Options that are available at this time are limited to the `SS_FORCE_TAPE_DISMOUNT` option. If this option is selected, tapes assigned to the session will be dismounted immediately, without waiting for another session to pick them up.

When `ss_EndSession` is called, the session handle is immediately invalidated, making all future references to the session impossible. The function then carries out a number of clean-up activities and returns when all server data structures related to the session have been deleted or otherwise processed.

4.1.3. `ss_GetStorageClassStats`

Purpose

Retrieve current information about storage classes managed by the server

Syntax

```
#include "ss_interface.h"

signed32
ss_GetStorageClassStats(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,           /* IN */
    signed32               ReqstID,        /* IN */
    ss_sclass_array_t     **Stats,        /* IN */
    trpc_status_t          *RPCStatus);    /* OUT */
```

Description

The `ss_GetStorageClassStats` function returns a list of storage classes managed by the server. Each element of the list contains information about the amount of storage space in the class and how much of it is free.

Parameters

| | |
|------------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>Stats</i> | A pointer to a pointer containing a variable length array containing the storage class statistics. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|---|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_EMMREAD | Metadata Manager failure |
| HPSS_ENOMEM | Server heap is exhausted. |
| HPSS_ESYSTEM | Encina transactional system failure. System log will contain more information about system failure. |

See also

None.

Clients

Storage System Manager.

Notes

This is a non-transactional function.

This function should be called with *Stats* pointing to a null pointer to a **ss_sclass_array_t** structure. The function builds a list of storage class statistics in an **ss_sclass_array_t** structure and points to it with **Stats*. The caller should not pass a structure to the function.

The **ss_sclass_array_t** structure contains a conformant array (variable length array) of **ss_sclass_t** elements, and a length field (number of elements in the conformant array).

When the caller has finished processing the **ss_sclass_array_t** structure, the memory space it occupies should be returned to the heap by calling "free" for each element of the list.

4.1.4. `ss_GetWaitingEvents`

Purpose

Retrieve a list of events the server is waiting on

Syntax

```
#include "ss_interface.h"

signed32
ss_GetWaitingEvents(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,           /* IN */
    signed32               ReqstID,        /* IN */
    ss_event_array_t      **Events,       /* IN */
    trpc_status_t          *RPCStatus);    /* OUT */
```

Description

The `ss_GetWaitingEvents` function returns a list of blocking events the server is waiting on. The possible events are pre-determined by the server and cannot be changed by a client.

Parameters

| | |
|------------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>Events</i> | A pointer to a pointer to an <code>ss_event_array_t</code> structure that contains the list of waiting events. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|----------------------------|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOMEM | Server heap is exhausted. |

See also

None.

Clients

Storage System Manager.

Notes

This is a non-transactional function. If the caller wants information about errors encountered by the RPC mechanism, an appropriate error variable should be added to the calling sequence with the `ss_interface.tacf` file.

This function should be called with *Events* pointing to a null pointer to an `ss_event_array_t` structure. The function builds a list of `ss_event_rec_t` elements in an `ss_event_array_t` structure and points to it with *Events*. The caller should not pass a structure to the function.

The **ss_event_array_t** structure contains a conformant array (variable length array) of **ss_event_rec_t** elements, and a length field (number of elements in the conformant array).

When the caller has finished processing the **ss_event_rec_t** elements, their memory space should be returned to the heap by calling "free" on **Events*.

4.1.5. **ss_MapCreate**

Purpose

Create a storage space map for a virtual volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_MapCreate(  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNH,             /* IN */  
    signed32               ReqstID,         /* IN */  
    hpssoid_t              *VVID,          /* IN */  
    u_signed64             NumBytes,       /* IN */  
    storage_class_id_t     SCID);         /* IN */
```

Description

The **ss_MapCreate** function creates a storage space map for the given virtual volume.

Parameters

| | |
|-----------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>VVID</i> | The object identifier of the virtual volume to create the map for. |
| <i>NumBytes</i> | The byte size of the virtual volume. |
| <i>SCID</i> | The Storage Class ID. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|--|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ENOENT | Virtual volume does not exist. |
| HPSS_EEXIST | Virtual volume already mapped. |
| HPSS_EPERM | Virtual volume has more than one owner. |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_EMMREAD | Error reading metadata file. |
| HPSS_EMMUPDATE | Error updating metadata file. |
| HPSS_EMMINSERT | Error inserting record into metadata file. |

See also

ss_MapDelete.

Clients

Storage System Manager.

Notes

Virtual volume must exist before the map is created.

One map per Virtual Volume.

NumBytes must equal the estimated Virtual Volume size.

A space map must be created for a Virtual Volume before Storage Segments can be created.

The disk Storage Server includes a bit map with the storage map that maps each virtual volume block on the disk.

4.1.6. **ss_MapDelete**

Purpose

Delete the storage space map for a virtual volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_MapDelete(  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNH,             /* IN */  
    signed32               ReqstID,         /* IN */  
    hpssoid_t              *VVID);         /* IN */
```

Description

The `ss_MapDelete` function deletes a storage space map for the specified virtual volume.

Parameters

| | |
|----------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>VVID</i> | The object identifier of the virtual volume to delete the map for. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|---|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Free space map does not exist. |
| HPSS_ENOMEM | Internal server memory failure. |
| HPSS_ECONFLICT | Map still has active segments. |
| HPSS_EMMREAD | Error reading metadata file. |
| HPSS_EMMUPDATE | Error updating metadata file. |
| HPSS_EMMDELETE | Error deleting a record from a metadata file. |

See also

ss_MapCreate.

Clients

Storage System Manager.

Notes

All storage segments must be deleted from the map before attempting to delete the map.

4.1.7. **ss_MapGetAttrs****Purpose**

Get the attributes of a virtual volume storage space map.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_MapGetAttrs(
    trpc_handle_t           Binding,           /* IN */
    hpss_connect_handle_t *CNH,             /* IN */
    signed32                ReqstID,         /* IN */
    hpssoid_t              *VVID,           /* IN */
    ss_map_attr_t          *MapAttrOut,     /* OUT */
    rpc_status_t            *RPCStatus);    /* OUT */
```

Description

The **ss_MapGetAttrs** function returns the attributes of a virtual volume storage map.

Parameters

| | |
|-------------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>VVID</i> | The object identifier of the virtual volume whose map is to be returned. |
| <i>MapAttrOut</i> | The returned storage map. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--------------------------------|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ENOENT | Free space map does not exist. |
| HPSS_EMMREAD | Error reading metadata file. |

See also

ss_MapSetAttrs.

Clients

Storage System Manager.

Notes

None.

4.1.8. **ss_MapSetAttrs**

Purpose

Set the attributes of a virtual volume storage space map.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_MapSetAttrs(  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNH,            /* IN */  
    signed32               ReqstID,         /* IN */  
    u_signed64             InSelectBitmap,   /* IN */  
    ss_map_attr_t         *InMapAttr,       /* IN */  
    u_signed64             *OutSelectBitmap, /* OUT */  
    ss_map_attr_t         *OutMapAttr);     /* OUT */
```

Description

The **ss_MapSetAttrs** function sets the attributes of a virtual volume storage map.

Parameters

| | |
|------------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>InSelectBitmap</i> | Bit flags that select the attribute(s) to change. |
| <i>InMapAttr</i> | A pointer to the storage map attribute record which contains the characteristics to set. |
| <i>OutSelectBitmap</i> | Bit flags that report attributes changed in <i>OutMapAttr</i> . |
| <i>OutMapAttr</i> | A pointer to the storage map attribute record which returns the characteristics of the storage map. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--------------------------------|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Free space map does not exist. |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EUPDATE | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager failure. |

See also

`ss_MapGetAttrs`.

Clients

Storage System Manager.

Notes

None.

4.1.9. **ss_PVCreate**

Purpose

Create a physical volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_PVCreate(  
    trpc_handle_t          Binding,      /* IN */  
    hpss_connect_handle_t *CNH,        /* IN */  
    signed32               ReqstID,     /* IN */  
    pv_attr_t              *PVAttrIn,   /* IN */  
    pv_attr_t              *PVAttrOut); /* OUT */
```

Description

The **ss_PVCreate** function creates a physical volume. The attributes of the new physical volume are input through a physical volume attribute record.

Parameters

| | |
|------------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>PVAttrIn</i> | A pointer to a physical volume attribute record which contains the characteristics of the physical volume to be created. |
| <i>PVAttrOut</i> | A pointer to a physical volume attribute record which contains the characteristics of the created physical volume. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|-------------------------------------|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | PVL does not know about the volume. |
| HPSS_EEXIST | Physical Volume already exists. |
| HPSS_EINVAL | Input parameter error. |
| HPSS_EFAULT | DCE or environment error. |
| HPSS_EOWNER | Bad owner attributes. |
| HPSS_EMMINSERT | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager error. |

See also

ss_PVDelete.

Clients

Storage System Manager.

Notes

A Physical volume entry is the building block for all other objects in the Storage Server. All other objects stored by the SS are built from Physical Volumes.

The PVL must have cataloged the physical volume before attempting to create it in the Storage Server.

Fields that may be set when creating a physical volume are listed in the **pv_attr_t** data structure definition.

4.1.10. **ss_PVDelete**

Purpose

Delete a physical volume.

Syntax

```
#include "ss_interface.h"

signed32
ss_PVDelete(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    signed32               ReqstID,         /* IN */
    char                   *PVName);       /* IN */
```

Description

The **ss_PVDelete** function deletes a physical volume.

Parameters

| | |
|----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>PVName</i> | The unique name of the physical volume. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|---|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | PVL does not know about the volume. |
| HPSS_EINUSE | Physical Volume is active doing I/O. |
| HPSS_EOWNER | Physical Volume is still owned by a Virtual Volume. |
| HPSS_EMMDELETE | Metadata Manager failure. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager error. |

See also

ss_PVCreate.

Clients

Storage System Manager.

Notes

The physical volume must not be pointed to by any virtual volume.

4.1.11. **ss_PVGetAttrs****Purpose**

Return the attributes of a physical volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_PVGetAttrs(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32              ReqstID,         /* IN */
    char                  *PVName,        /* IN */
    pv_attr_t             *PVRetAttr,     /* OUT */
    trpc_status_t         *RPCStatus);    /* OUT */
```

Description

The **ss_PVGetAttrs** function returns the attributes of a physical volume.

Parameters

| | |
|------------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>PVName</i> | The unique name of the physical volume to delete, maximum of 8 characters. |
| <i>PVRetAttr</i> | A pointer to a physical volume attribute record which contains the characteristics of the physical volume. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Physical volume record does not exist. |
| HPSS_EMMREAD | Metadata Manager failure. |

See also

ss_PVSetAttrs.

Clients

Storage System Manager, Virtual Volume layer.

Notes

This functions is non-transactional.

4.1.12. **ss_PVMount****Purpose**

Mount one or more physical volumes.

Syntax

```
#include "ss_interface.h"

signed32
ss_PVMount(
    trpc_handle_t          Binding,      /* IN */
    hpss_connect_handle_t *CNH,        /* IN */
    hpss_object_handle_t  *Session,    /* IN */
    signed32               ReqstID,     /* IN */
    pv_list_t              *PVListIn,   /* IN */
    pv_list_t              *PVListOut); /* OUT */
```

Description

The **ss_PVMount** function mounts one or more physical volumes.

Parameters

| | |
|------------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>PVListIn</i> | A list of physical volumes to mount. |
| <i>PVListOut</i> | The list of physical volumes mounted, and the associated information for each mount. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Physical Volume does not exist. |
| HPSS_ENOMOUNT | Trying to set position on a Physical Volume that is not mounted. |
| HPSS_ESYSTEM | Transaction manager failure. |

See also

ss_PVUnmount.

Clients

Virtual volume layer, Storage System Manager.

Notes

Mount requests that contain a list of PVs to mount are assumed to be atomic, i.e. they are to be mounted as a single mount job. If a group of PVs is to be mounted individually, then a call to **ss_PVMount** must be made for each PV.

In a disk Storage Server, all disks are mounted when the server starts. Calls to **ss_PVMount** cause no physical mount to occur and return immediately.

4.1.13. **ss_PVRead****Purpose**

Read one or more physical volumes.

Syntax

```
#include "ss_interface.h"

signed32
ss_PVRead(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32               ReqstID,         /* IN */
    IOD_t                  *PVIODPtr,      /* IN */
    IOR_t                  *PVIORPtr);     /* OUT */
```

Description

The **ss_PVRead** function issues reads to a Mover, wait for replies, and then reply to the client with the IOR.

Parameters

| | |
|-----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>PVIODPtr</i> | A pointer to the structure which describes the I/O requests. |
| <i>PVIORPtr</i> | A pointer to the structure which describes the state of I/O requests at completion of the transfer. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|---|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOMOUNT | Physical Volume is not mounted. |
| HPSS_ECONFLICT | Physical Volume being accessed by other thread. |
| HPSS_EINVAL | IOD was invalid or inconsistent. |
| HPSS_EFAULT | DCE or environment error. |
| HPSS_ENOMEM | Server exhausted internal memory. |
| HPSS_EPERM | Volume protected against reads. |
| HPSS_EMMREAD | Metadata Manager failure. |

| | |
|----------------|--|
| HPSS_EMMUPDATE | Metadata Manager failure. |
| HPSS_EOM | Trying to read off end of volume. |
| HPSS_ESYSTEM | Transaction manager failure. |
| other | Data movement errors may occur during a read. These errors are described in the IOD/IOR document.(EOM, EIO, etc.). |

See also

HPSS Programmer's Reference, Volume 1 for the format and use of IODs and IORs.

ss_PVWrite.

Clients

Virtual Volume layer, Storage System Manager.

Notes

PVs must be mounted before issuing this function.

If the result of the read is zero(0), the IOR must still be checked to verify what data was actually moved and if any errors occurred.

4.1.14. **ss_PVSetAttrs****Purpose**

Set the attributes of a physical volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_PVSetAttrs(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32              ReqstID,         /* IN */
    u_signed64            InSelectBitmap   /* IN */
    pv_attr_t             *InPVAttr,       /* IN */
    u_signed64            *OutSelectBitmap, /* OUT */
    pv_attr_t             *OutPVAttr);     /* OUT */
```

Description

The **ss_PVSetAttrs** function sets the attributes of a physical volume as specified through the input attribute record.

Parameters

| | |
|------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>Select</i> | Bit flags that select the attribute(s) to change. |
| <i>PVAttrIn</i> | A pointer to a physical volume attribute record which contains the characteristics of the physical volume to set. The physical volume is selected by setting the Name field in this record. |
| <i>Selected</i> | Bit flags that report attributes changed in <i>PVAttrOut</i> . |
| <i>PVAttrOut</i> | A pointer to a physical volume attribute record which contains the characteristics of the physical volume after the set. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ENOENT | Physical Volume does not exist. |
| HPSS_EOWNER | could not set <i>VVID</i> field in Physical Volume. |
| HPSS_ENOMOUNT | trying to set position on a Physical Volume that is not mounted. |

| | |
|----------------|---|
| HPSS_EINVAL | Input parameter error. |
| HPSS_ECONFLICT | Physical Volume being accessed by other thread. |
| HPSS_EFAULT | DCE or environment error. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EMMUPDATE | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager error. |

See also

ss_PVGetAttrs.

Clients

Storage System Manager.

Notes

Physical Volume must be mounted to set Current Position.

A Physical volume must be mounted to change the position characteristics.

Changes to position characteristics or the ending of a section will have no effect on disk positioning or media format.

4.1.15. **ss_PVUnmount****Purpose**

Unmount one or more physical volumes.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_PVUnmount(  
    trpc_handle_t          Binding,      /* IN */  
    hpss_connect_handle_t *CNH,        /* IN */  
    hpss_object_handle_t  *Session,    /* IN */  
    signed32               ReqstID,     /* IN */  
    pv_list_t              *PVList);   /* IN */
```

Description

The **ss_PVUnmount** function unmounts one or more physical volumes.

Parameters

| | |
|----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>PVList</i> | A list of physical volumes to unmount. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--------------------------------------|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Physical Volume in list not mounted. |

See also

ss_PVMount.

Clients

Physical volume layer, Virtual volume layer, Storage System Manager.

Notes

The unmount will not cause a physical unmount to fixed media and will return without error.

4.1.16. `ss_PVWrite`

Purpose

Write a physical volume.

Syntax

```
#include "ss_interface.h"

signed32
ss_PVWrite(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32               ReqstID,         /* IN */
    IOD_t                  *PVIODPtr,      /* IN */
    IOR_t                  *PVIORPtr);     /* OUT */
```

Description

The `ss_PVWrite` function issues writes to a set of Mover's, waits for replies, and then replies to the client with an IOR.

Parameters

| | |
|-----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>PVIODPtr</i> | A pointer to the structure which describes the I/O requests. |
| <i>PVIORPtr</i> | A pointer to the structure which describes the state of I/O requests at completion of the transfer. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|--|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Physical Volume does not exist. |
| HPSS_ENOMOUNT | trying to set position on a Physical Volume that is not mounted. |
| HPSS_EINVAL | Input parameter error. |
| HPSS_EFAULT | DCE or environment failure. |
| HPSS_ENOMEM | Internal server memory exhausted. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EMMUPDATE | Metadata Manager failure. |

HPSS_EOM Physical end of media was reached.

HPSS_ESYSTEM Transaction manager failure.

See also

HPSS Programmer's Reference, Volume 1 for the format and use of IODs and IORs.

ss_PVRead.

Clients

Virtual Volume layer, Storage System Manager.

Notes

PVs must be mounted before issuing this function.

If the result is zero(0) the IOR must still be checked to verify what data actually was moved and if any errors occurred during data movement.

4.1.17. **ss_ServerGetAttrs**

Purpose

Gets the generic attributes of the Storage Server.

Syntax

```
#include "ss_interface.h"

signed32
ss_ServerGetAttrs(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    signed32               ReqstID,         /* IN */
    hpss_server_attr_t     *SSAttrOut,     /* OUT */
    trpc_status_t          *RPCStatus);     /* OUT */
```

Description

The **ss_ServerGetAttrs** function returns the generic attributes of the Storage Server.

Parameters

| | |
|------------------|---|
| <i>Binding</i> | TRPC structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SSAttrOut</i> | A pointer to a Storage Server attribute record which returns the characteristics of the Storage Server. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|----------------------------|
| HPSS_EBADCONN | Connect handle is invalid. |
| HPSS_EINVAL | Input parameter invalid |

See also

ss_ServerSetAttrs.

Clients

Storage System Manager.

Notes

This is a non-transactional function.

4.1.18. **ss_ServerSetAttrs****Purpose**

Change the generic characteristics of the Storage Server.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_ServerSetAttrs(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,           /* IN */
    signed32               ReqstID,        /* IN */
    u_signed64             InSelectBitmap,  /* IN */
    hpss_server_attr_t    *InSSAttr,      /* IN */
    u_signed64             OutSelectBitmap, /* OUT */
    hpss_server_attr_t    *OutSSAttr,     /* OUT */
    trpc_status_t         *RPCStatus);    /* OUT */
```

Description

The **ss_ServerSetAttrs** function changes the attributes of the Storage Server.

Parameters

| | |
|------------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>InSelectBitmap</i> | Bit flags that select the attribute(s) to change. |
| <i>InSSAttr</i> | A pointer to a Storage Server attribute record which contains the characteristics to set. |
| <i>OutSelectBitmap</i> | Bit flags that report attributes changed in <i>OutSSAttr</i> . |
| <i>OutSSAttr</i> | A pointer to a Storage Server attribute record which returns the characteristics of the Storage Server. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|----------------------------|
| HPSS_EBADCONN | Connect handle is invalid. |
| HPSS_EINVAL | Input parameter invalid. |

See also

ss_ServerGetAttrs.

Clients

Storage System Manager.

Notes

This is a non-transactional function.

Only the *RegisterBitmap* and *AdministrativeState* fields of the server state can be changed with this function.

4.1.19. **ss_SSCopySegment****Purpose**

Copy the data content of a storage segment to a new storage segment on a different virtual volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSCopySegment(
    trpc_handle_t           Binding,           /* IN */
    hpss_connect_handle_t  *CNH,             /* IN */
    hpss_object_handle_t   *Session,         /* IN */
    signed32                ReqstID,         /* IN */
    hpssoid_t              *SrcSSID,         /* IN */
    hpssoid_t              *VVID,           /* IN */
    ss_attr_t               *SSAttrOut,      /* OUT */
    trpc_status_t          *RPCStatus);      /* OUT */
```

Description

The **ss_SSCopySegment** function creates a new storage segment using the attributes of the source segment on a different virtual volume, and copies the source segment's data content to the new segment.

Parameters

| | |
|------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SrcSSID</i> | The object identifier of the storage segment that contains the source data. |
| <i>VVID</i> | The object identifier of the virtual volume that should be used, if possible, as the destination. If this argument is zero, it is ignored and any available virtual volume is used. |
| <i>SSAttrOut</i> | The attributes of the new segment. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|---------------------------------------|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Storage Segment does not exist. |
| HPSS_ENOMOUNT | Storage Segment could not be mounted. |

| | |
|----------------|---|
| HPSS_EPOSERROR | Virtual Volume could not be positioned to the beginning of the segment.. |
| HPSS_EFAULT | DCE or internal system error. |
| HPSS_ENOMEM | Internal server memory exhausted. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_ENOSPACE | There is no space to create the destination segment. |
| HPSS_EINVAL | <i>VVID</i> is invalid or points to an inappropriate virtual volume (non-matching storage class). |

See also

ss_SSMoveSegment.

Clients

Migration and Caching Clients, Replication Clients, Storage System Manager.

Notes

This function is non-transactional.

The source segment does not need to be mounted prior to executing this function. If the source segment is mounted in another session, this function will block until the volume is freed.

If *VVID* points to a valid virtual volume, **ss_SSCopySegment** will attempt to use the volume as the destination. If the volume does not contain sufficient space a different volume is used. If the volume parameters are inappropriate, HPSS_EINVAL is returned.

4.1.20. **ss_SSCreate****Purpose**

Create a storage segment.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSCreate(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32               ReqstID,        /* IN */
    unsigned32             FamilyId,       /* IN */
    unsigned32             Options,        /* IN */
    signed32               Count,          /* IN */
    ss_attr_t              *SSAttrIn,     /* IN */
    ss_attr_t              *SSAttrOut,    /* OUT */
    ss_segment_array_t    **Segments);    /* OUT */
```

Description

The **ss_SSCreate** function creates one or more storage segments. The attributes for the new storage segments are input through a storage segment attribute record, *SSAttrIn*. The attributes of the created storage segment are returned in a second storage segment attribute record, *SSAttrOut*. The number of segments to create, options for the operation and the *FamilyId* to assign to the segments are input through the arguments. The storage segment SOIDs are returned through an output conformant array (*Segments*).

Parameters

| | |
|-----------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | A pointer to a session handle that identifies a session to be associated with the segment. See notes below. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>FamilyId</i> | The identifier for the family in which the segment is to be created. This argument is supported by the tape storage server only. A value of zero means that the segment must be created on volumes not associated with a family. |
| <i>Options</i> | Option flags that control the details of the operation. The flag <code>SS_CREATE_HINT_MANDATORY</code> causes the storage server to return an error (<code>HPSS_ENOSPACE</code>) if the segments cannot be created on the VV described in the SSID hint. If the flag is not given, another VV will be chosen. |
| <i>Count</i> | The number of segments to create. The tape storage server can create only one segment for each call to ss_SSCreate and will return an error (<code>HPSS_EINVAL</code>) if this argument is any other value. The disk storage server can create any number of |

| | |
|------------------|---|
| | segments for each call to ss_SSCreate and requires only that Count be greater than zero. |
| <i>SSAttrIn</i> | A pointer to a storage segment attribute record which contains the characteristics requested for the storage segment to be created. |
| <i>SSAttrOut</i> | A pointer to a storage segment attribute record which contains the characteristics of the storage segment created. |
| <i>Segments</i> | A pointer to a pointer to a <code>ss_segment_array_t</code> structure. The caller should preset <i>*Segments</i> to null. The server will create a conformant array of SOIDs in the <code>ss_segment_array_t</code> structure and return a pointer to it in <i>*Segments</i> .. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|---|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ECONFLICT | <i>Session</i> is already is use. |
| HPSS_ENOSPACE | No space available. |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_EOWNER | Invalid or empty owner set in attributes. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EMMINSERT | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager failure. |

See also

None.

Clients

Bitfile Server.

Notes

Fields that may be set when creating a storage segment are listed in the **ss_attr_t** data structure definition.

Tape and disk storage segment creation follow different rules that are complex. They are outlined separately below:

Tape storage segment creation:

The tape storage server can create only one storage segment for each call to **ss_SSCreate**. This limitation is imposed by the nature of sequential magnetic tape, i.e., the tape can be written only at its logical end, so tape storage segments must be created created, written and terminated sequentially.

The tape Storage Server recognizes two hints for selection of a virtual volume for a new tape storage segment. If the *SSAttrIn->MD.SSID* field is non-zero, it is taken to be the ID of an existing tape storage segment. The server will attempt to create a new segment on the same virtual volume that contains *SSAttrIn->MD.SSID*. If the selected volume is assigned to another session, the session reservation is overridden by this option. If no space is left in that volume, or the volume is already busy, the Storage Server will create the segment on another volume.

The session parameter is used as a virtual volume hint as well. If a tape storage segment has previously been created in the session given in the call, the virtual volume used for that segment will be tried for the new segment. If the volume is available and has the correct storage class and sufficient free space, the segment will be created. If these criteria cannot be met, other volumes are tried. If a segment is successfully created, its virtual volume will be assigned to the session.

Once a tape storage segment is created, the storage map for the virtual volume becomes busy and stays busy until the session is closed, or the storage segment is terminated (see **ss_SSWrite**).

Disk storage segment creation:

The algorithm for creating disk storage segments is different from the tape algorithm in some important ways.

The disk storage server can create multiple storage segments for each call to **ss_SSCreate**. It selects a disk Virtual Volume and creates as many segments on it as possible, up to the value of "Count". The characteristics of the segments (block size, allocated length, etc.) are all alike in the batch of segments.

The SS ID hint, described above, is employed, if provided by the caller, but the session hint is not available. The session argument in disk storage segment creation calls is included only for consistency with the tape calling sequence.

In tape storage segments, the *AllocatedLength* parameter is considered by the server to be a guide to the length of the segment, but more or less information can be written to the segment, depending on the available space on the virtual tape.

In disk storage segments, the *AllocatedLength* parameter sets the maximum length of the disk segment. The data area of the segment can be written at any location, in any order, but data cannot be written past the *AllocatedLength*. The segment can be shortened (see **ss_SSSetAttrs**).

Segments on disk are allocated at their declared size, rounded up to the next multiple of the VV block size. The segment is allocated contiguous blocks of the underlying disk virtual volume. The segment length can be decreased (disk blocks are given up), and can be increased only if the number of disk blocks making up the segment does not increase. No provision is made for adding disk blocks to storage segments at this time.

FamilyId is not supported by the disk storage server. Disk storage segments do not have a meaningful family id attribute.

Disk storage maps are locked while the transaction in which the **ss_SSCreate** call is made remains unresolved. When the transaction resolves, the storage map becomes available for a new segment creation operation immediately. The map does not stay busy as it does for tape storage segments.

4.1.21. `ss_SSDelete`

Purpose

Delete a storage segment.

Syntax

```
#include "ss_interface.h"
```

```
signed32
ss_SSDelete(
trpc_handle_t          Binding,      /* IN */
hpss_connect_handle_t *CNH,        /* IN */
signed32              ReqstID,     /* IN */
hpssoid_t             SSID)        /* IN */
```

Description

The `ss_SSDelete` function deletes a storage segment.

Parameters

| | |
|----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SSID</i> | The SOID of the segment to be deleted. |

Return Values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error Conditions

| | |
|---------------|---|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ENOENT | Segment does not exist |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_EOWNER | The owner attributes are invalid. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EUPDATE | Metadata Manager failure. |
| HPSS_EDELETE | Segment could not be deleted from metadata. |
| HPSS_ESYSTEM | Transaction system error. |

Related Information

None.

Clients

BitFile Server.

Notes

The space occupied by tape storage segments cannot be reused until the virtual volume is reclaimed. The space occupied by disk storage segments can be reused immediately after `ss_SSUnlink` finishes.

4.1.22. `ss_SSDeleteList`

Purpose

Delete a list of storage segments.

Syntax

```
#include "ss_interface.h"

signed32
ss_SSDelete(
trpc_handle_t          Binding,          /* IN */
hpss_connect_handle_t *CNH,            /* IN */
signed32               ReqstID          /* IN */
ss_delete_segment_array_t *Segments)   /* IN/OUT*/
```

Description

The `ss_SSDeleteList` function deletes a list storage segments.

Parameters

| | |
|----------|--|
| Binding | DCE structure used to locate a server on the network. |
| CNH | Used to maintain server specific state for a particular client. |
| ReqstID | User supplied request identifier. |
| Segments | A pointer to a <code>ss_delete_segment_array_t</code> structure that contains a conformant array of storage segment SOIDs and error codes. |

Return Values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error Conditions

| | |
|----------------|--|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ENOTREADY | Server initialization not complete, or server shutting down. |
| HPSS_ENOENT | Segment does not exist |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EUPDATE | Metadata Manager failure. |
| HPSS_EDELETE | Segment could not be deleted from metadata. |
| HPSS_ESYSTEM | Transaction system error. |

Related Information

`ss_SSDelete`.

Clients

BitFile Server.

Notes

The space occupied by tape storage segments cannot be reused until the virtual volume is reclaimed. The space occupied by disk storage segments can be reused immediately after `ss_SSUnlink` finishes.

Most errors are returned on a per storage segment basis, in the error codes in the “Segments” argument.

`ss_SSDeleteList` is a non-transactional function.

4.1.23. **ss_SSGetAttrs**

Purpose

Gets the attributes of a storage segment.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSGetAttrs(  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNH,            /* IN */  
    signed32               ReqstID,         /* IN */  
    hpssoid_t              *SegID,         /* IN */  
    ss_attr_t              *SSAttrOut,     /* OUT */  
    trpc_status_t          *RPCStatus);    /* OUT */
```

Description

ss_SSGetAttributes returns the attributes of a storage segment.

Parameters

| | |
|------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SegID</i> | The object identifier of the storage segment to operate on. |
| <i>SSAttrOut</i> | A pointer to a storage segment attribute record which returns the characteristics of the storage segment. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|-------------------------------|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ENOENT | Segment does not exist. |
| HPSS_EMMREAD | Metadata Manager failure. |

See also

ss_SSSetAttrs.

Clients

Bitfile Server, Storage System Manager.

Notes

This is a non-transactional function.

4.1.24. **ss_SSMount****Purpose**

Mount a segment and assign it to a session.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSMount(
    trpc_handle_t          Binding,      /* IN */
    hpss_connect_handle_t *CNH,        /* IN */
    hpss_object_handle_t  *Session,    /* IN */
    signed32               ReqstID,    /* IN */
    hpssoid_t              *SSID);     /* IN */
```

Description

The virtual volume associated with the named storage segment is located and reserved to the given session. An **ss_VVMount** function is then performed and the volume is mounted and positioned to the start of the storage segment. The automatic dismount of idle storage segments is disabled.

Parameters

| | |
|----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SSID</i> | The object identifier of the segment to mount. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|--|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Storage Segment does not exist. |
| HPSS_ENOMOUNT | Storage Segment could not be mounted. |
| HPSS_EPOSERROR | Virtual Volume could not be positioned to the beginning of the segment.. |
| HPSS_EFAULT | DCE or internal system error. |
| HPSS_ENOMEM | Internal server memory exhausted. |
| HPSS_EMMREAD | Metadata Manager failure. |

See also

ss_SSUnmount.

Clients

Migration and Caching Clients, Repacking Clients, Storage System Manager.

Notes

The **ss_SSMount** function is provided to optimize tape-to-tape copying of storage segments. The function allows a copy facility to mount the source and sink tapes in advance of starting a copy operation.

In disk Storage Servers, this function causes the mounting of the appropriate unmounted disk volume. If the volume is mounted, the function returns immediately.

4.1.25. **ss_SSMoveSegment****Purpose**

Move a storage segment to a new virtual volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSMoveSegment(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32              ReqstID,         /* IN */
    hpssoid_t             *SSID,           /* IN */
    hpssoid_t             *VVID,           /* IN */
    ss_attr_t             *SSAttrOut,      /* OUT */
    trpc_status_t         *RPCStatus);     /* OUT */
```

Description

ss_SSMoveSegment creates a new data body for the given segment and copies the data from the original virtual volume to the new virtual volume. When successful, the original data body is deleted. The segment's identifier, *SSID*, does not change.

Parameters

| | |
|------------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SSID</i> | The object identifier of the storage segment. |
| <i>VVID</i> | Where to move the segment. If the <i>VVID</i> is all zero's then a Virtual Volume with the same characteristics as the segment will be chosen. |
| <i>SSAttrOut</i> | The updated attributes of the moved segment. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--|
| HPSS_EBADCONN | Connect handle is invalid. |
| HPSS_ENOENT | Storage Segment is not currently mounted. |
| HPSS_ENOSPACE | There is no space to move the data. |
| HPSS_EINVAL | <i>VVID</i> is unavailable or inappropriate. |

See also

ss_SSCopySegment

Clients

Migration and Caching Clients, Repacking Clients, Maintenance utilities, Storage System Manager.

Notes

This function is non-transactional.

The source segment does not need to be mounted prior to executing this function. If the source segment is mounted in another session, this function will block until the volume is freed.

If *VVID* points to a valid virtual volume, **ss_SSMoveSegment** will attempt to use the volume as the destination. If the volume does not contain sufficient space a different volume is used. If the volume parameters are inappropriate, `HPSS_EINVAL` is returned.

It is best to call **ss_SSMoveSegment** with a session that has not been used for any other purpose. This reduces the possibility of the server having difficulty reserving the necessary resources.

4.1.26. **ss_SSRead****Purpose**

Read data from one or more storage segments.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSRead(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32               ReqstID,         /* IN */
    IOD_t                  *SSIODPtr,      /* IN */
    IOR_t                  *SSIORPtr);     /* OUT */
```

Description

The **ss_SSRead** function reads data from the storage segments given in *SSIODPTR* to the destination given in *SSIODPTR*. An IOR is filled in with the results of the operation.

Parameters

| | |
|-----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | Identifies the Storage Server session in which to perform the function. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SSIODPtr</i> | A pointer to the structure which describes the I/O requests. |
| <i>SSIORPtr</i> | A pointer to the structure which describes the state of I/O requests at completion of the transfer. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned. In the case of a zero (0) error being returned, the IOR must still be checked to see what data was moved. . If the function completed without error, but an error was reported in the IOR from elsewhere in the system, the function returns with the IOR status value.

Error conditions

| | |
|----------------|---|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Storage segment does not exist. |
| HPSS_ENOMOUNT | Could not mount the virtual volume that contains the storage segment. |
| HPSS_ECONFLICT | Storage segment is busy in some other thread. |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_EMMREAD | Metadata Manager failure. |

| | |
|----------------|---|
| HPSS_EMMUPDATE | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager failure. |
| other: | Error moving data. Error translates to data movement error. |

See also

ss_SSWrite, ss_BeginSession.

Clients

Bitfile Server.

Notes

Tape virtual volumes that are the targets of the read operation are mounted and unmounted as needed by this function.

The read function is synchronous - the function blocks until the read operation is complete, with or without error, before returning the IOR. Disk differs from tape in that many disk I/O's may be active to a particular storage media, while on tape only one I/O may be active at a time. The server guarantees this behavior.

4.1.27. **ss_SSrvGetAttrs****Purpose**

Gets the attributes of the Storage Server.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSrvGetAttrs(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    unsigned32             ReqstID,         /* IN */
    ssrv_attr_t            *SSAttrOut,     /* OUT */
    trpc_status_t          *RPCStatus);    /* OUT */
```

Description

The **ss_SSrvGetAttrs** function returns the attributes of the Storage Server.

Parameters

| | |
|------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SSAttrOut</i> | A pointer to a Storage Server attribute record which returns the characteristics of the Storage Server. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|-------------------------------|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_EMMREAD | Metadata Manager failure. |

See also

ss_SSrvSetAttrs.

Clients

Storage System Manager.

Notes

This is a non-transactional function.

4.1.28. **ss_SSrvSetAttrs**

Purpose

Change the characteristics of the Storage Server.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSrvSetAttrs(  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNH,            /* IN */  
    signed32               ReqstID,         /* IN */  
    u_signed64             InSelectBitmap,   /* IN */  
    ssrv_attr_t            *InSSAttr,       /* IN */  
    u_signed64             *OutSelectBitmap, /* OUT */  
    ssrv_attr_t            *OutSSAttr,      /* OUT */  
    trpc_status_t          *RPCStatus);     /* OUT */
```

Description

The **ss_SSrvSetAttrs** function changes the attributes of the Storage Server.

Parameters

| | |
|------------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>OutSelectBitmap</i> | Bit flags that select the attribute(s) to change. |
| <i>InSSAttr</i> | A pointer to a Storage Server attribute record which contains the characteristics to set. |
| <i>InSelectBitmap</i> | Bit flags that report attributes changed in <i>OutSSAttr</i> . |
| <i>OutSSAttr</i> | A pointer to a Storage Server attribute record which returns the characteristics of the Storage Server. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|----------------------------|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EMMUPDATE | Metadata Manager failure. |

HPSS_ESYSTEM

Encina transactional system failure. System log will contain more information about system failure.

See also

ss_SSrvGetAttrs.

Clients

Storage System Manager.

Notes

This is a non-transactional function.

4.1.29. **ss_SSSetAttrs**

Purpose

Change the characteristics of a storage segment

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSSetAttrs(  
    trpc_handle_t          Binding,           /* IN */  
    hpss_connect_handle_t *CNH,             /* IN */  
    hpss_object_handle_t  *Session,         /* IN */  
    signed32              ReqstID,          /* IN */  
    u_signed64            InSelectBitmap,    /* IN */  
    ss_attr_t             *InSSAttr,        /* IN */  
    u_signed64            *OutSelectBitmap, /* OUT */  
    ss_attr_t             *OutSSAttr);      /* OUT */
```

Description

The **ss_SSSetAttrs** function changes the attributes of a storage segment.

Parameters

| | |
|------------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | Identifies the Storage Server session in which to perform the function. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>InSelectBitmap</i> | Bit flags that select the attribute(s) to change. |
| <i>InSSAttr</i> | A pointer to a storage segment attribute record which contains the characteristics to set. The storage segment is selected by setting the <i>SSID</i> field in this record. |
| <i>OutSelectBitmap</i> | Bit flags that report attributes changed in <i>OutSSAttr</i> . |
| <i>OutSSAttr</i> | A pointer to a storage segment attribute record which returns the new characteristics of the storage segment. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|---------------------------------|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Storage segment does not exist. |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_ECONFLICT | Segment busy performing I/O. |

| | |
|----------------|------------------------------|
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EMMUPDATE | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager failure. |

See also

ss_SSGetAttrs.

Clients

Bitfile Server, Storage System Manager.

Notes

The fields in **ss_attr_t** that may be changed by this function are listed in the **ss_attr_t** data structure definition.

Attributes to be set are indicated with bit flags in the *SelectFlags* argument. Certain attributes may not be changed.

Owner information is set or deleted according to the value of the DELETE_OWNER flag in the calling **owner_rec_t** structure passed in as part of *InSSAttr*. If DELETE_OWNER is set, a matching owner record will be deleted from the storage segment records. If not set, the owner will be added.

Changes to positioning information or attempts to force media marks on disk will succeed but will have no effect on the media (e.g. no real mark will be written and addressing will not be altered).

In the tape Storage Server, setting the size of the *AllocatedLength* is meaningless. In the disk server changing *AllocatedLength* is meaningful. If the new length is shorter than the current length the vacated space is returned to the storage map for reassignment (segment truncation). If the new length is greater than the current length, an attempt is made to increase the size of the storage segment by appending space to the segment (segment extension). This can be done only if the space following the segment is available and long enough. If no space is available, an error is returned. If some space is available, but not as much as requested, it is appended and the new (less than expected) length is returned.

In the disk server, *WrittenLength* is meaningless. Setting *WrittenLength* is allowed, but has no effect

4.1.30. `ss_SSStartMount`

Purpose

Start a distributed mount of a storage segment

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSSetAttrs(  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNH,            /* IN */  
    hpss_object_handle_t  *Session,        /* IN */  
    signed32               *Active,         /* IN */  
    signed32               ReqstID,         /* IN */  
    signed32               *SSID,          /* IN */  
    hpssoid_t              *NewJob,         /* IN/OUT */  
    signed32               *PVLJobID);     /* IN/OUT */
```

Description

The virtual volume associated with the name storage segment is located and reserved to the given session. A mount operation is started for the VV, but is completed at some later time. If *Active* is True, a new mount job is started and *PVLJobId* is set to the new PVL job id. If *Active* is false, the VV to be mounted is added to an existing PVL job given by *PVLJobId*.

If *Active* is True, *NewJob* is an output argument and will be True if a new PVL job was started. If *Active* is False, *NewJob* is an input argument that is True if the active side of the distributed mount started a new job.

Parameters

| | |
|-----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | Identifies the Storage Server session in which to perform the function. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>Active</i> | True if this function is called by the active side of a distributed mount; false if called by the passive side. |
| <i>SSID</i> | The object identifier of the segment to mount. |
| <i>NewJob</i> | <i>NewJob</i> works as described above. |
| <i>PVLJobId</i> | The PVL job id of the new PVL job. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|----------------------------|
| HPSS_EBADCONN | Invalid connection handle. |
|---------------|----------------------------|

| | |
|----------------|---|
| HPSS_ENOENT | Storage segment does not exist. |
| HPSS_ENOMOUNT | Storage Segment could not be mounted. |
| HPSS_EPOSERROR | Virtual Volume could not be positioned to the beginning of the segment. |
| HPSS_EFAULT | DCE or internal system error. |
| HPSS_ENOMEM | Internal server memory exhausted. |
| HPSS_EMMREAD | Metadata Manager failure. |

See also

ss_SSRead, ss_SSWRITE.

Clients

Migration and Caching Clients, Repacking Clients, SSM.

Notes

This function is used by clients to mount the Vvs for two storage segments, in a single PVL mount job. Using a single job prevents the possibility of mount deadlocks that can result from use two PVL jobs.

The two storage segments are expected to be part of a segment to segment copy operation, usually disk segment to tape segment. The caller must divide the operation into active and passive parts. The tape segment usually takes the active role and the disk segment usually takes the passive role.

The call to **ss_SSStartMount** on the *Active* side comes first. When the call returns, a PVL job id will have been assigned, but the VV will not yet be mounted. Next, the call to the passive side is made using the PVL job id returned from the active side. The passive side call adds its VV to the mount job and returns.

The caller may then go ahead and call **ss_SSRead** or **ss_SSWrite** to perform the I/O. If the mount job has not completed at the time these functions are called, the mount is waited for.

4.1.31. **ss_SSUnlink**

Purpose

Unlink a storage segment.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSUnlink(
```

```
    trpc_handle_t          Binding,          /* IN */
```

```
    hpss_connect_handle_t *CNH,          /* IN */
```

```
    signed32              ReqstID,          /* IN */
```

```
    ss_attr_t             *SSAttrIn);      /* IN */
```

Description

The **ss_SSUnlink** function unlinks a storage segment. The reference count for the storage segment is decremented and if the result is zero, the segment is deleted from the server.

Parameters

| | |
|-----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SSAttrIn</i> | Attributes that describe the segment to be unlinked. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|---|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ENOENT | Segment does not exist. |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_EOWNER | The owner attributes are invalid. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EUPDATE | Metadata Manager failure. |
| HPSS_EDELETE | Segment could not be deleted from metadata. |
| HPSS_ESYSTEM | Transaction system error. |

See also

None.

Clients

Bitfile Server.

Notes

The space occupied by tape storage segments cannot be reused until the virtual volume is reclaimed. The space occupied by disk storage segments can be reused immediately after **ss_SSUnlink** finishes.

4.1.32. **ss_SSUnmount**

Purpose

Unmount a storage segment.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSUnmount(  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNH,             /* IN */  
    hpss_object_handle_t  *Session,         /* IN */  
    signed32               ReqstID,         /* IN */  
    hpssoid_t              *SSID);         /* IN */
```

Description

The **ss_SSUnmount** function unmounts a storage segment. The associated virtual volume is unmounted and disassociated from the session.

Parameters

| | |
|----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SSID</i> | The object identifier of the storage segment. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|---|
| HPSS_EBADCONN | Connect handle is invalid. |
| HPSS_ENOENT | Storage Segment is not currently mounted. |

See also

ss_SSMount.

Clients

Migration and Caching Clients, Repacking Clients, Storage System Manager.

Notes

In the disk Storage Server, virtual volumes are never actually unmounted. The logically mounted volume is unmounted (dissociated from the session).

4.1.33. **ss_SSWrite****Purpose**

Write data to one or more storage segments.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_SSWrite(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32               ReqstID,        /* IN */
    IOD_t                  *SSIODPtr,      /* IN */
    IOR_t                  *SSIORPtr);     /* OUT */
```

Description

The **ss_SSWrite** function writes data to the storage segments given in *SSIODPTR* to the destination given in *SSIODPTR*. An IOR is filled in with the results of the operation.

Parameters

| | |
|-----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | Identifies the Storage Server session in which to perform the function. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>SSIODPtr</i> | A pointer to the structure which describes the I/O requests. |
| <i>SSIORPtr</i> | A pointer to the structure which describes the state of I/O requests at completion of the transfer. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|---|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Storage segment does not exist. |
| HPSS_ENOMOUNT | Could not mount the virtual volume that contains the storage segment. |
| HPSS_ECONFLICT | Storage segment is busy in some other thread. |
| HPSS_EINVAL | Input parameter invalid. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EMMUPDATE | Metadata Manager failure. |

| | |
|--------------|---|
| HPSS_ESYSTEM | Transaction manager failure. |
| HPSS_EOM | End of media has been reached. Segment can not be extended further. |
| other: | Error moving data. Error translates to data movement error. |

See also

ss_SSRead, ss_BeginSession.

Clients

Bitfile Server.

Notes

Tape virtual volumes that are the targets of the write operation are mounted and unmounted as needed by this function.

The write function is synchronous - the function blocks until the write operation is complete, with or without error, before returning the IOR. Disk and tape write operations look the same to the caller, however long delays associated with mounting tapes are to be expected.

The semantics of disk storage segment writes are different from tape segment writes. Tape storage segments can only be appended to. Disk segment writes can occur anywhere within the limits of the segment, but segments cannot be extended by writing past the end of the segment. Other than insuring that disk storage segment writes occur entirely within the limits of the segment, the disk Storage Server does not record any information about which portions of disk storage segments have been written.

There is no EOM condition on segments. Attempts to write outside of the address space of a disk storage segment receive an HPSS_EINVAL error.

4.1.34. **ss_VVCreate****Purpose**

Create a virtual volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_VVCreate(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    signed32               ReqstID,         /* IN */
    vv_attr_t              *VVAttrIn,       /* IN */
    vv_attr_t              *VVAttrOut,      /* OUT */
    hpssoid_t              *VVID);          /* OUT */
```

Description

The **ss_VVCreate** function creates a new virtual volume. The attributes of the new virtual volume are input through a virtual volume attribute record.

Parameters

| | |
|------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>VVAttrIn</i> | A pointer to a virtual volume attribute record which contains the characteristics requested for the virtual volume to be created. |
| <i>VVAttrOut</i> | A pointer to a virtual volume attribute record which contains the characteristics of the virtual volume created. |
| <i>VVID</i> | The object identifier of the created virtual volume. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|---|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | At least one Physical Volume does not exist. |
| HPSS_EEXIST | Physical Volume already part of another Virtual Volume. |
| HPSS_EINVAL | Input parameter error. |
| HPSS_ENOMEM | Server memory exhausted. |
| HPSS_EFAULT | Internal DCE or system library error. |
| HPSS_EMMREAD | Metadata Manager failure. |

| | |
|----------------|----------------------------|
| HPSS_EMMINSERT | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager error. |

See also

`ss_VVDelete`.

Clients

Storage Server layer, Storage System Manager.

Notes

Fields that may be set when creating a virtual volume are listed in the `vv_attr_t` data structure definition.

The disk virtual volume will be mounted and the metadata cached during this operation.

4.1.35. **ss_VVDelete****Purpose**

Delete a virtual volume

Syntax**#include "ss_interface.h"****signed32****ss_VVDelete(**

| | | |
|------------------------------|-----------------|-----------------|
| trpc_handle_t | <i>Binding,</i> | <i>/* IN */</i> |
| hpss_connect_handle_t | <i>*CNH,</i> | <i>/* IN */</i> |
| signed32 | <i>ReqstID,</i> | <i>/* IN */</i> |
| hpssoid_t | <i>*VVID);</i> | <i>/* IN */</i> |

Description

The `ss_VVDelete` function deletes a virtual volume. A virtual volume must have no owners when deleted.

Parameters

| | |
|----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>VVID</i> | The object identifier of the virtual volume to delete. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|---|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Virtual Volume does not exist. |
| HPSS_EEINUSE | Virtual Volume is currently being accessed. |
| HPSS_EOWNER | Virtual Volume is owned and being used. |
| HPSS_ENOMEM | Server exhausted memory. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EMMDELETE | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager error. |

See also**ss_VVCreate.****Clients**

Storage Server layer, Storage System Manager.

Notes

None.

4.1.36. **ss_VVGetAttrs****Purpose**

Gets the attributes of a virtual volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_VVGetAttrs(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32               ReqstID,         /* IN */
    hpssoid_t              *VVID,          /* IN */
    vv_attr_t              *VVAttrOut,     /* OUT */
    trpc_status_t          *RPCStatus);    /* OUT */
```

Description

The **ss_VVGetAttributes** function returns the attributes of a virtual volume record.

Parameters

| | |
|------------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>VVID</i> | The object identifier of the virtual volume. |
| <i>VVAttrOut</i> | A pointer to a virtual volume attribute record which contains the characteristics of the virtual volume. |
| <i>RPCStatus</i> | A pointer to a returned RPC status code. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|-----------------------------------|
| HPSS_EBADCONN | Connection handle is invalid. |
| HPSS_ENOENT | Virtual Volume does not exist. |
| HPSS_ENOMEM | Internal server memory exhausted. |
| HPSS_EMMREAD | Metadata Manager failure. |

See also

ss_VVSetAttrs.

Clients

Storage Server layer, Storage System Manager.

Notes

This function is non-transactional.

Attributes that represent the position of the media will not be valid for a disk Storage Server and will be set to zero's (0).

4.1.37. **ss_VVMount****Purpose**

Mount a virtual volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_VVMount(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,           /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32              ReqstID,         /* IN */
    hpssoid_t            *VVID,           /* IN */
    composite_address_t  *StartAbsAddr,    /* IN */
    relative_address_t   *StartRelAddr,    /* IN */
    relative_address_t   *DesiredRelAddr); /* IN */
```

Description

The **ss_VVMount** function gets the Virtual Volume mounted and positioned to the desired location.

Parameters

| | |
|-----------------------|--|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>VVID</i> | The object identifier of the virtual volume to mount. |
| <i>StartAbsAddr</i> | The media specific absolute addressing information of the start of a block on the virtual volume.(used for tape only). |
| <i>StartRelAddr</i> | The relative address of the start address of a block on the virtual volume.(used for tape only). |
| <i>DesiredRelAddr</i> | The desired byte position within the block given by the start address.(used for tape only). |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|---|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Virtual Volume does not exist. |
| HPSS_ENOMOUNT | Virtual Volume could not be mounted. |
| HPSS_EPOSERROR | Virtual Volume could not be positioned. |
| HPSS_EFAULT | DCE or internal system error. |

| | |
|--------------|-----------------------------------|
| HPSS_ENOMEM | Internal server memory exhausted. |
| HPSS_EMMREAD | Metadata Manager failure. |

See also

ss_VVUnmount.

Clients

Storage Server Layer, Storage System Manager.

Notes

The positioning information will not be used for disk, it will only have meaning for tape. For consistency however; the client may wish to store and pass this information at all times, so that the client does not have to distinguish between a disk and a tape Storage Server.

4.1.38. **ss_VVRead****Purpose**

Read data from a virtual volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_VVRead(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32               ReqstID,        /* IN */
    IOD_t                  *VVIODPtr,      /* IN */
    IOR_t                  *VVIORPtr)      /* OUT */
```

Description

The **ss_VVRead** function translates IOD virtual volume addresses into physical volume addresses and issue reads and wait for replies from the Physical Volume layer, then replies to the caller with the IOR.

Parameters

| | |
|-----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>VVIODPtr</i> | A pointer to the structure which describes the I/O requests. |
| <i>VVIORPtr</i> | A pointer to the structure which describes the state of I/O requests at completion of the transfer. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Virtual Volume does not exist. |
| HPSS_ENOMOUNT | Virtual Volume could not be mounted. |
| HPSS_EINVAL | Input parameter error. |
| HPSS_EPERM | Virtual Volume state does not allow reads. |
| HPSS_ENOMEM | Internal server memory exhausted. |
| HPSS_EMMREAD | Metadata Manager failure. |

| | |
|----------------|---|
| HPSS_EMMUPDATE | Metadata Manager failure. |
| HPSS_EFAULT | DCE or other environment error. |
| HPSS_EOM | Trying to read off the end of media. |
| HPSS_ESYSTEM | Transaction manager error. |
| other | Errors that may occur during any data movement operation. |

See also

ss_VVWrite.

Clients

Storage Server layer, Storage System Manager.

Notes

IODs may refer to more than one virtual volume.

If the result is zero(0), the IOR must still be checked to verify what data was actually moved and any errors that may have occurred during the move.

4.1.39. **ss_VVSetAttrs****Purpose**

Sets the attributes of a virtual volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_VVSetAttrs(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32               ReqstID,        /* IN */
    u_signed64             InSelectBitmap,  /* IN */
    vv_attr_t              *InVVAttr,      /* IN */
    u_signed64             OutSelectBitmap, /* OUT */
    vv_attr_t              *OutVVAttr);    /* OUT */
```

Description

The **ss_VVSetAttrs** function changes the attributes of a virtual volume record.

Parameters

| | |
|------------------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>InSelectBitmap</i> | Bit flags that select the attribute(s) to change. |
| <i>InVVAttr</i> | A pointer to a virtual volume attribute record which contains the characteristics requested for the virtual volume. The virtual volume is selected by setting the <i>VVID</i> field in this record. |
| <i>OutSelectBitmap</i> | Bit flags that report attributes changed in <i>VVAttrOut</i> . |
| <i>OutVVAttr</i> | A pointer to a virtual volume attribute record which contains the characteristics of the virtual volume. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|---|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Virtual Volume does not exist |
| HPSS_ENOMOUNT | Virtual Volume could not be mounted. |
| HPSS_EPOSERROR | Virtual Volume could not be positioned. |

| | |
|----------------|------------------------------|
| HPSS_EOWNER | Could not delete owner. |
| HPSS_EINVAL | Input parameter error. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EMMUPDATE | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager failure. |

See also

ss_VVGetAttrs.

Clients

Storage Server layer, Storage System Manager.

Notes

The fields in **vv_attr_t** that may be changed by this function are listed in the **vv_attr_t** data structure definition.

The virtual volume must be mounted to set the *CurrentPosition*.

Owner information is set or deleted according to the value of the DELETE_OWNER flag in the calling **owner_rec_t** structure passed in as part of *SSAttrIn*. If DELETE_OWNER is set, a matching owner record will be deleted from the storage segment records. If not set, the owner will be added.

Setting the position information or an attempt to end section on a disk virtual volume will succeed but will cause no I/O to the disk media and will not change the Section field of the address as it does on tape.

4.1.40. **ss_VVUnmount****Purpose**

Unmount a virtual volume.

Syntax

```
#include "ss_interface.h"
```

```
signed32
```

```
ss_VVUnmount(
    trpc_handle_t          Binding,      /* IN */
    hpss_connect_handle_t *CNH,        /* IN */
    hpss_object_handle_t  *Session,    /* IN */
    signed32               ReqstID,    /* IN */
    hpssoid_t              *VVID);     /* IN */
```

Description

The **ss_VVUnmount** function unmounts a virtual volume.

Parameters

| | |
|----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>VVID</i> | The object identifier of the virtual volume. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--|
| HPSS_EBADCONN | Connect handle is invalid. |
| HPSS_ENOENT | Virtual volume is not currently mounted. |

See also

ss_VVMount.

Clients

Storage Server Layer, Virtual Volume Layer, Storage System Manager.

Notes

The unmount command will not actually issue an unmount to the PVL in the disk Storage Server.

4.1.41. **ss_VVWrite**

Purpose

Write data to a virtual volume.

Syntax

```
#include "ss_interface.h"

signed32
ss_VVWrite(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    hpss_object_handle_t  *Session,        /* IN */
    signed32               ReqstID,        /* IN */
    IOD_t                  *VVIOD,         /* IN */
    IOR_t                  *VVIOR);        /* OUT */
```

Description

The **ss_VVWrite** function translates IOD virtual volume addresses into physical volume addresses and issue writes and waits for replies from the Physical Volume layer, then replies to the client with the IOR.

Parameters

| | |
|----------------|---|
| <i>Binding</i> | DCE structure used to locate a server on the network. |
| <i>CNH</i> | Used to maintain server specific state for a particular client. |
| <i>Session</i> | The handle which identifies the session. |
| <i>ReqstID</i> | User supplied request identifier. |
| <i>VVIOD</i> | A pointer to the structure which describes the I/O requests. |
| <i>VVIOR</i> | A pointer to the structure which describes the state of I/O requests at completion of the transfer. |

Return values

Upon successful completion the function returns a zero(0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--------------------------------------|
| HPSS_EBADCONN | Invalid connection handle. |
| HPSS_ENOENT | Virtual Volume does not exist |
| HPSS_ENOMOUNT | Virtual Volume could not be mounted. |
| HPSS_EINVAL | Input parameter error. |
| HPSS_EAPPEND | Tape not at append point. |
| HPSS_EOM | End of tape was hit on the write. |
| HPSS_EPERM | VV state does not allow writes. |

| | |
|----------------|---|
| HPSS_ENOMEM | Internal server memory exhausted. |
| HPSS_EFAULT | DCE or environment error. |
| HPSS_EMMREAD | Metadata Manager failure. |
| HPSS_EMMUPDATE | Metadata Manager failure. |
| HPSS_ESYSTEM | Transaction manager error. |
| other | Errors that may occur during data movement. See IOD/IOR document. |

See also

ss_VVRead.

Clients

Storage Server layer, Storage System Manager.

Notes

IODs may refer to more than one virtual volume.

If the result of a write call is zero(0), the IOR must still be checked to verify what data was actually moved and if any errors occurred during the data movement.

4.2. Data Definitions

This section describes key internal data definitions and all externally used data definitions which are provided by this subsystem. A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

4.2.1. **Storage Server Attribute Record - `ssrv_attr_t`**

Description

The `ssrv_attr_t` is used to transfer information about the Storage Server across Storage Server functional interfaces.

Format

The `ssrv_attr_t` has the following format:

```
typedef struct ssrv_attr {
    uuid_t      ServerID;
    u_signed64  RegisterBitMap;
    u_signed64  TotalVirtualVolumes
    u_signed64  TotalAllocatedVolumes
    u_signed64  TotalBytes;
    u_signed64  UsedBytes;
    u_signed64  FreeBytes;
    char        SS_MAP_MetaData_Name[HPSS_MAX_DCE_NAME];
    char        SS_SS_MetaData_Name[HPSS_MAX_DCE_NAME];
    char        SS_VV_MetaData_Name[HPSS_MAX_DCE_NAME];
    char        SS_PV_MetaData_Name[HPSS_MAX_DCE_NAME];
    char        SS_SC_MetaData_Name[HPSS_MAX_DCE_NAME];
    signed32    ReadDrives;
    signed32    WriteDrives;
    uuid_t      PVL_UUID;
} ssrv_attr_t;
```

The following fields can be modified with a `ss_SSrvSetAttrs` function call:

| | | |
|----------------|---------------------|-----------------------|
| RegisterBitMap | TotalVirtualVolumes | TotalAllocatedVolumes |
| TotalBytes | UsedBytes | FreeBytes |

The remaining fields in the attribute record are readable but not writeable.

ServerID

Unique identifier for the server.

RegisterBitMap

Map of registered `ssrv_attr_t` fields.

TotalVirtualVolumes

The total number of Virtual Volumes supported by this server.

TotalAllocatedVolumes

The total number of Virtual Volumes on which storage space has been allocated.

TotalBytes

The total number of bytes of storage available to this Storage Server. These bytes are available through

the Storage Segment interface.

UsedBytes

The number of bytes that have been recorded. This field applies to the Storage Segment interface.

FreeBytes

The number of unused bytes available for recording. This field applies to the Storage Segment interface.

SS MAP MetaData Name

Path name of the file that contains the metadata that describes the Volume Maps. Volume Maps are described later in this document.

SS SS MetaData Name

Path name of the file that contains the metadata that describes the Storage Segments. Storage Segments are described later in this document.

SS VV MetaData Name

Path name of the file that contains the metadata that describes the Virtual Volumes. Virtual Volumes are described later in this document.

SS PV MetaData Name

Path name of the file that contains the metadata that describes Physical Volumes. Physical Volumes are described later in this document.

SS SC MetaData Name

Path name of the file that contains the system storage class metadata.

ReadDrives

The number of drives that are available for reading Physical Volumes.

WriteDrives

The number of drives that are available for writing Physical Volumes.

PVL UUID

Unique identifier of the Physical Volume Library. This UUID is used to validate connections made to the Storage Server.

Clients

The following clients access the data definition:

Storage System Manager.

4.2.2. Storage Segment Record - storage_segment_record_t

Description

A storage_segment_record is a volatile record that describes a Storage Segment. It is used to hold information needed to control access to storage segments with a session. This record is an internal structure within the Storage Server and is used to keep track of active segments within the server.

Format

The storage_segment_record has the following format:

```
typedef struct storage_segment_record {
```

```
hpssoid_t          SSID;
hpssoid_t          VVID;
unsigned32         Flags;
signed32           ThreadsUsing;
u_signed64        WrittenLength;
hps_connect_handle_t ConnectionHandle;
hps_object_handle_t SessionHandle;
pthread_cond_t     Cond;
timestamp_sec_t   TimeStamp;
tran_tid_t        OwingTid;
tran_tid_t        CreateTid;
tran_tid_t        DeleteTid;
storage_segment_md_t *MD;
struct storage_segment_record *next, *prev;
struct storage_segment_record *newer, *older;
struct storage_segment_record *next_sess_ssr,
                              *prev_sess_ssr;
struct storage_segment_record *next_list;
} storage_segment_record_t;
```

SSID

Storage segment ID.

VVID

Virtual Volume that contains the segment.

Flags

| | |
|----------------------|---|
| SS_IO | Segment has I/O in progress. |
| SS_DELETED | Record is no longer valid. |
| SS_WRITTEN | Segment has been written. |
| SS_INUSE | Cache entry in use. |
| SS_DESTROYED | Cache entry to be removed. |
| SS_VOIDMD | Cache MD must be reloaded. |
| SS_BADMD | Cache MD must be reloaded once more. |
| SS_DIRTY | MD points to metadata that needs to be flushed to SFS. |
| SS_READ_WITHOUT_LOCK | Cache MD was last read without using a bracket read lock (possibly a dirty read). |

ConnectionHandle

The connection on which the Storage Segment was accessed.

SessionHandle

Session associated with this record.

Cond

Condition variable to wait for locks on the structure (not used in tape Storage Servers).

TimeStamp

Time record last used.

OwningTid, CreateTid, DeleteTid

Transaction IDs that control use of the cache entry.

MD

Pointer to metadata associated with this record (not used in tape Storage Servers). The condition of the MD (valid or invalid) is determined by the state of the Flags field.

next, prev

Active segment chain links.

Newer, older

Links to newer and older records in cache.

next_sess_ssr, prev_sess_ssr

Active session chain links.

next_list

Call-back chain link.

Clients

The following clients access the data definition:

Storage Server.

4.2.3. Storage Segment Attribute Record - ss_attr_t

Description

The ss_attr_t is used to transfer information about storage segments across Storage Server functional interfaces.

Format

The ss_attr_t has the following format:

```
typedef struct ss_attr {
    storage_segment_md_t MD;
    u_signed64 RegisterBitMap;
} ss_attr_t;
```

The following segment metadata fields can be written during the process of creating a storage segment, but remain fixed afterwards:

| | | |
|-------------------|------|-------------------|
| SClassId | VVID | AbsoluteStartAddr |
| RelativeStartAddr | | |

The following segment metadata fields can be modified with a ss_SSSetAttributes function call:

| | | |
|------------------|------------------|---------------------|
| Acct | Allocated_Length | WrittenLength |
| OperationalState | UsageState | AdministrativeState |
| SSState | RegisterBitMap | Owners |

The remaining fields in the attribute record are readable but not writeable.

MD

The storage segment metadata record.

RegisterBitMap

The map of registered notification fields.

Clients

The following clients access the data definition:

Bitfile Server, Storage System Manager.

4.2.4. **Storage Segment Metadata - storage_segment_md_t**

Description

A storage segment metadata record is a permanent record that describes a storage segment.

Format

The storage_segment_md has the following format:

```
typedef struct storage_segment_md {
    hpssoid_t          SSID;
    hpssoid_t          VVID;
    acct_rec_t         Acct;
    unsigned32         SClassId;
    signed32           RefCnt;
    signed32           NumReads;
    signed32           NumWrites;
    signed32           OperationalState;
    signed32           UsageState;
    signed32           AdministrativeState;
    signed32           SSState;
    unsigned32         BlockSize;
    u_signed64         AllocatedLength;
    u_signed64         WrittenLength;
    composite_address_t AbsoluteStartAddr;
    relative_address_t  RelativeStartAddr;
    relative_address_t  RelativeNextByteAddr;
    owner_rec_t        Owners;
    security_t         Security;
    timestamp_sec_t    LastRead;
    timestamp_sec_t    LastWrite;
    timestamp_sec_t    Creation;
    timestamp_sec_t    Update;
    unsigned32         Unused[8];
} storage_segment_md_t;
```

SSID

The storage segment ID. Primary search key.

VVID

The associated virtual volume ID. Secondary search key.

Acct

Accounting information.

SClassId

Storage Class ID values as defined in the bit file server.

RefCnt

Number of references to this segment. If this number is zero then the segment may be deleted.

NumReads

Total number of reads.

NumWrites

Total number of writes.

OperationalState

Operational state flags:

ST_ENABLED

ST_DISABLED

UsageState

Usage state flags:

ST_IDLE

ST_ACTIVE

ST_BUSY

AdministrativeState

Administrative state flags:

ST_LOCKED

ST_UNLOCKED

SSState

SEG_NULL Segment is invalid and does not map any space.

SEG_ALLOCATED Segment maps space and allows appends.

SEG_LENGTH_FIXED Segment mapped but no appends are allowed.

BlockSize

The block size of the media.

AllocatedLength

The allocated length of the segment, in bytes.

WrittenLength

The address of the last byte written into the segment, plus one. This field is not meaningful in disk Storage Servers and is ignored.

AbsoluteStartAddr

Concatenated media absolute addresses that define the starting point of the storage segment on the media. This field is not meaningful in disk Storage Servers and is ignored.

RelativeStartAddr

The relative address that defines the starting point of the storage segment on the media. This address is relative to the beginning of the media. This field is not meaningful in disk Storage Servers and is ignored.

RelativeNextByteAddr

The relative address that defines the byte following the ending point of the storage segment. This address is used for an append operation and is relative to the beginning of the media. This field is not meaningful in disk Storage Servers and is ignored.

Owners

Owner is an owner_rec_t that defines the owner of this storage segment. Each client wishing to be linked with this storage segment provides a non-zero SOID that can be used by the Storage Server to contact the client. Zero entries in the array are vacant. Each time a non-zero entry is added to the array, RefCnt is incremented by one.

Security

Security information of unknown format.

LastRead

The time of last read.

LastWrite

The time of last write.

Creation

The time the segment was created.

Update

The time of the last change to the segment

Clients

The following clients access the data definition:

Bitfile server, Storage System Manager

4.2.5. **Storage Map Record - storage_map_record_t**

Description

A storage_map_record_t is a volatile record that describes a Storage Map. It is used to hold information needed to control access to storage maps. This record is an internal structure within the Storage Server and is used to keep track of active maps within the server.

Format

The storage_map_record has the following format:

```
typedef struct storage_map_record {
    hpsoid_t          VVID;
    unsigned32       Flags;
    pthread_cond_t   Cond;
```

```

signed32          ThreadsUsing;
tran_tid_t       OwingTid;
tran_tid_t       CreateTid;
tran_tid_t       DeleteTid;
u_signed64       PrevSpaceLeft;
storage_map_md_t *MD;
storage_map_md_t *OldMD;
signed32         CondVar;
struct storage_map_record *next_map, *prev_map;
struct storage_map_record *next_search, *prev_search;
} storage_map_record_t;

```

VVID

Virtual Volume that is mapped by the record.

Flags

| | |
|-----------------------|---|
| MAP_INUSE | Cache entry in use. |
| MAP_DESTROYED | Cache entry to be deleted. |
| MAP_VOIDMD | Cache MD must be reloaded. |
| MAP_BADMD | Cache MD must be reloaded once. |
| MAP_READ_WITHOUT_LOCK | Cache MD was last read without using a bracket read lock (possibly a dirty read). |

Cond

Condition variable to wait for locks on the structure (not used in tape Storage Servers).

ThreadsUsing

Count of the number of threads using or waiting for the cache entry.

OwingTid, CreateTid, DeleteTid

Transaction IDs that control use of the cache entry.

PrevSpaceLeft

Previous value of SpaceLeft. Used to keep cache consistent.

MD

Pointer to metadata associated with this record (not used in tape Storage Servers). The condition of the MD (valid or invalid) is determined by the state of the Flags field.

OldMD

Pointer to a metadata record containing the state of the metadata prior to any changes. This record is compared with "MD" to determine which fields have changed, so that only those fields will be changed in the metadata record on disk.

Next map, prev map

Active segment chain links.

Next search, prev search

Storage map search links. The disk storage maps are linked together using these links to form a loop which the server uses to determine the next storage map to search for storage space.

Clients

The following clients access the data definition:

Storage Server.

4.2.6. Storage Map Attribute Record - `ss_map_attr_t`

Description

The `ss_map_attr_t` is used to transfer information about storage maps across Storage Server functional interfaces.

Format

The `ss_map_attr_t` has the following format:

```
typedef struct ss_map_attr {
    storage_map_md_t  MD;
    u_signed64       RegisterBitMap;
} ss_map_attr_t;
```

The following map metadata fields can be written during the process of creating a storage map, but remain fixed afterwards:

| | |
|----------|------|
| SClassId | VVID |
|----------|------|

The following map metadata fields can be modified with a `ss_MapSetAttributes` function call:

| | | |
|---------------------|------------|----------------|
| OperationalState | UsageState | |
| AdministrativeState | MapState | RegisterBitMap |
| TapeSz | EstSz | SpaceLeft |

The remaining fields in the attribute record are readable but not writeable.

MD

The storage map metadata record.

RegisterBitMap

The map of registered notification fields.

Clients

The following clients access the data definition:

Bitfile Server, Storage System Manager

4.2.7. Tape Storage Map Metadata - `storage_map_md_t`

Description

The `storage_map_md_t` is a permanent record that describes the usage of the storage space controlled by the Storage Server. There is one `storage_map_md` associated with each tape virtual volume. Storage segments are created by allocating space from a `storage_map_md`, and assigning it to the storage segments.

Format

The `storage_map_md_t` has the following format:

```
typedef struct storage_map_md {
    hpsoid_t    VVID;
    hpsoid_t    CurrentSSID;
    signed32    NumActiveSegments;
    signed32    OperationalState;
    signed32    UsageState;
    signed32    AdministrativeState;
    signed32    MapState;
    unsigned32  SClassId;
    unsigned32  Flags;
    unsigned32  BlkSz;
    u_signed64  TapeSz;
    u_signed64  EstSz;
    unsigned32  FamilyId;
    unsigned32  Unused[7];
} storage_map_md_t;
```

VVID

The associated virtual volume ID. Primary search key.

CurrentSSID

The ID of the current segment that is being written. This field used only with append type media (Tape, Optical, etc.).

NumActiveSegments

The number of storage segments associated with this map.

OperationalState

Operational state flags:

ST_ENABLED

ST_DISABLED

UsageState

Usage state flags:

ST_IDLE

ST_ACTIVE

ST_BUSY

ST_UNKNOWN

AdministrativeState

Administrative state flags:

ST_LOCKED

ST_UNLOCKED

MapState

MAP_NULL The map is invalid and maps no real space.

| | |
|---------------|--|
| MAP_FREE | Map allows space allocation. |
| MAP_ALLOCATED | Map has free space, but the last segment created is still being written. (This state only valid on append type media). |
| MAP_EOM | No more free space in map. This value is used only in tape storage maps. |
| MAP_EMPTY | Tape has reached EOM and all storage segments have been removed. |
| MAP_TALLY | Storage map is a special storage class tally map. |
| MAP_RETIRED | Tape has been retired. Once retired, the state cannot be changed. Storage space cannot be created on retired volumes. |

SClassId

Storage Class ID values as defined in the bit file server. This field is a secondary search key along with SpaceLeft.

Flags

| | |
|-------------------|--|
| MAP_NEVER_WRITTEN | True until first allocation taken from map. |
| MAP_STAT_FREE | True when the VV is counted as FREE in the storage class statistics. |
| MAP_STAT_PARTIAL | True when the VV is counted as partially filled in the storage class statistics. |

BlkSz

The block size in bytes of the media associated with this storage_map_md. This is the minimum size that can be written atomically to the media.

TapeSz

Contains the number of bytes currently allocated on the media

EstSz

Estimated size is a constant byte size set when the map is initialized and is an estimate of the capacity of the virtual volume.

SpaceLeft

SpaceLeft is initialized to the value of EstSz and is decremented as the space is consumed. This field is a secondary search key along with SClassId. This field should be considered a hint or guide and will not be exact.

FamilyId

The family associated with the tape volume. If zero, no family is associated.

Clients

The following clients access the data definition:

SS Layer, Storage System Manager.

4.2.8. Disk Storage Map Metadata

Description

The `disk_storage_map_md_t` is a permanent record that describes the usage of the disk storage space controlled by the Storage Server. There is one `disk_storage_map_md` associated with each disk virtual volume. Storage segments are created by allocating space from a `disk_storage_map_md`, and assigning it to the storage segments.

Format

The `disk_storage_map_md_t` has the following format:

```
typedef struct disk_storage_map_md {
    hpssoid_t          VVID;
    hpssoid_t          CurrentSSID;
    signed32           NumActiveSegments;
    signed32           OperationalState;
    signed32           UsageState;
    signed32           AdministrativeState;
    signed32           MapState;
    unsigned32         SClassId;
    unsigned32         Flags;
    unsigned32         BlkSz;
    u_signed64         TapeSz;
    u_signed64         EstSz;
    u_signed64         SpaceLeft;
    unsigned32         FamilyId;
    unsigned32         Unused[7];
    byte               BitMap[MAX_ALLOC_BITMAP_SIZE];
} disk_storage_map_md_t;
```

VVID

The associated virtual volume id. Primary search key.

CurrentSSID

This field is not used. It is provided to keep `disk_storage_map_md_t` congruent with `storage_map_md_t`.

NumActiveSegments

The number of storage segments associated with this map.

OperationalState

Operational state flags:

ST_ENABLED

ST_DISABLED

UsageState

Usage state flags

ST_IDLE

ST_ACTIVE

ST_BUSY

ST_UNKNOWN

AdministrativeState

Administrative state flags

ST_LOCKED

ST_UNLOCKED

MapState

| | |
|-------------|--|
| MAP_NULL | The map is invalid and maps no real space. |
| MAP_FREE | Map allows space allocation. |
| MAP_RETIRED | Disk has been retired. Once retired the state cannot be changed. Storage space cannot be created on retired volumes. |

SClassId

Storage Class ID values as defined in the bit file server. This field is a secondary search key along with SpaceLeft.

Flags

| | |
|-------------|--------------------------------------|
| MAP_OFFLINE | True when map describes VV off-line. |
|-------------|--------------------------------------|

BlkSz

The block size in bytes of the media associated with this storage_map_md. This is the minimum size that can be written atomically to the media.

TapeSz

Contains the number of bytes currently allocated on the media. This field is set to the exact size of the disk and will be equal to the 'EstSz' field.

EstSz

Estimated size is a constant byte size set when the map is initialized. This field gives the exact size of the disk.

SpaceLeft

SpaceLeft is initialized to the value of EstSz and is decremented as the space is consumed. This field is a secondary search key along with SClassId. On disks, this number will be exact.

BitMap

Bit map that keeps track of used space on the disk. Each bit in the map designates the state of a VV block of disk storage. On the SFS disk, this field is represented by an array of byte arrays. Each of these sub-fields is updated only if it changes to minimize the amount of information transmitted to SFS and to minimize the SFS transaction log.

Clients

The following clients access the data definition:

SSLayer, SSM.

4.2.9. **Virtual Volume Record - virtual_volume_record_t**

Description

A `virtual_volume_record` is a volatile record that describes a virtual volume. This record is only active and valid when the volume is mounted.

Format

The `virtual_volume_record` has the following format:

```
typedef struct virtual_volume_record {
    hpssoid_t                VVID;
    unsigned32               Flags;
    pthread_cond_t           Cond;
    signed32                 ThreadsUsing;
    signed32                 LastState;
    tran_tid_t               OwningTid;
    tran_tid_t               CreateTid;
    tran_tid_t               DeleteTid;
    relative_address_t       CurrentRelativeAddr;
    composite_address_t      CurrentAbsoluteAddr;
    relative_address_t       NextRelativeAddr;
    composite_address_t      NextAbsoluteAddr;
    hpss_connect_handle_t    ConnectionHandle;
    hpss_object_handle_t     SessionHandle;
    pv_list_t                *PVLlist;
    virtual_volume_md_t       *MD;
    struct virtual_volume_record *SharedRecordPtr;
    struct virtual_volume_record *next_active, *prev_active;
    struct virtual_volume_record *next_sess, *prev_sess;
    signed32                 NumActiveSessions;
    signed32                 VolumeState;
    signed32                 AdministrativeState;
    unsigned32               StripeWidth;
    u_signed64               NumSectionBytes;
    u_signed64               BlockSize;
    u_signed64               StripeLength;
} virtual_volume_record_t;
```

VVID

ID of the volume.

Flags

| | |
|----------------------|---|
| VV_INUSE | Record busy in cache. |
| VV_DESTROYED | Record will be removed from cache. |
| VV_DIRTY | Record needs to be written to disk. |
| VV_BADMD | Record needs to be reloaded from disk. |
| VV_MOUNT_SUCCESS | VV mounted successfully. |
| VV_READ_WITHOUT_LOCK | Cache MD was last read without using a bracket read lock (possibly a dirty read). |
| VV_REREAD_ON_FINISH | Cache entry to be reloaded on commit. |

Cond

Condition variable for blocking access to the structure.

ThreadsUsing

Count of the number of threads using or waiting for the cache entry.

LastState

Used for cache consistency.

OwningTid, CreateTid, DeleteTid

Transaction IDs that control use of the cache entry.

CurrentRelativeAddr

Address of the append position of the media (not used in tape Storage Servers). This field is only used for append type media. This address is relative to the beginning of the Virtual Volume. This field is updated as a write proceeds.

CurrentAbsoluteAddr

Address of the append position of the media (not used in tape Storage Servers). This field is only valid for append type media. This address is absolute and will only be valid on media that supports absolute addressing. This field is updated as a write proceeds.

NextRelativeAddr

Address of the append position after the last successful write (not used in tape Storage Servers). This field will only be used for append type media. This field will only be updated at the end of an append type operation. This address is relative to the beginning of the media.

NextAbsoluteAddr

Address of the append position after the last successful write (not used in tape Storage Servers). This field will only be used for append type media. This field will only be updated at the end of an append type operation. This address is only valid on media that supports absolute addressing modes.

ConnectionHandle

The ID of the connection on who's behalf the Virtual Volume is mounted.

SessionHandle

The ID to the session on who's behalf the Virtual Volume is mounted.

PVList

Pointer to the list of Physical Volumes that make up the Virtual Volume.

MD

Pointer to cached Virtual Volume metadata. This field is currently not used on tape. If not NULL and virtual volume is a disk, then a pointer to valid metadata.

SharedRecordPtr

Points to a shared virtual_volume_record that contains mount information (disk only).

next_active, prev_active

Active segment chain links.

next_sess, prev_sess

Active session chain links.

NumActiveSessions

The number of sessions that desire this Virtual Volume to be mounted.

VolumeState

See VolumeState in virtual_volume_md.

AdministrativeState

Administrative state flags:

ST_LOCKED

ST_UNLOCKED

StripeWidth

The number of PV's that make up this VV.

NumSectionBytes

The number of bytes to be written to the media between marks.

BlockSize

The block size of the Virtual Volume.

StripeLength

The number of bytes that make a full stripe across the VV.

Clients

The following clients access the data definition:

Storage Server.

4.2.10. Virtual Volume Attribute Record - vv_attr_t**Description**

The vv_attr_t is used to transfer information about virtual volumes across Storage Server functional interfaces.

Format

The vv_attr_t has the following format:

```
typedef struct vv_attr {
    virtual_volume_md_t    MD
    u_signed64             RegisterBitMap;
    pv_list_t              *PVList;
    relative_address_t     CurrentRelativeAddr;
    composite_address_t    CurrentAbsoluteAddr;
} vv_attr_t;
```

The following fields can be written during the process of creating a virtual volume, but remain fixed afterwards:

| | | |
|-------------|--------------|-------|
| SClassId | Form | BlkSz |
| StripeWidth | StripeLength | |

The following fields can be modified with a ss_VVSetAttributes function call:

| | | |
|------|---------------|------------|
| Acct | EstimatedSize | ActualSize |
|------|---------------|------------|

| | | |
|---------------------|----------------|----------|
| CurrentRelativeAddr | Flags | Security |
| CurrentAbsoluteAddr | RegisterBitMap | |

The remaining fields in the attribute record are readable but not writeable.

MD

The virtual volume metadata record.

RegisterBitMap

The map of registered notification fields.

PVList

List of associated physical volumes that make up the Virtual Volume.

CurrentRelativeAddr

Relative address of current position of volume (not used in disk Storage Servers). This field is only valid for mounted Virtual Volumes.

CurrentAbsoluteAddr

Absolute address of current position of volume (not used in disk Storage Servers). This field is only valid for mounted Virtual Volumes.

Clients

The following clients access the data definition:

Bitfile Server, Storage System Manager.

4.2.11. **Virtual Volume Metadata - virtual_volume_md_t**

Description

A virtual volume metadata record describes the characteristics of a virtual volume.

Format

The virtual_volume_md has the following format:

```
typedef struct virtual_volume_md {
    hpssoid_t          VVID;
    acct_rec_t         Acct;
    unsigned32         SClassId;
    unsigned32         Form;
    unsigned32         BlkSz;
    unsigned32         StripeWidth;
    unsigned32         Flags;
    signed32           RefCnt;
    signed32           NumReads;
    signed32           NumWrites;
    signed32           OperationalState;
    signed32           UsageState;
    signed32           AdministrativeState;
    signed32           VVState;
    u_signed64         StripeLength;
    u_signed64         EstimatedSize;
    u_signed64         ActualSize;
    relative_address_t NextByteAddr;
    composite_address_t NextAbsoluteAddr;
    owner_rec_t        Owners;
```

```

security_t          Security;
timestamp_sec_t    LastRead;
timestamp_sec_t    LastWrite;
timestamp_sec_t    Creation;
timestamp_sec_t    Update;
unsigned32         FamilyId;
unsigned32         Unused[7];
} virtual_volume_md_t;

```

VVID

The virtual volume ID. Primary search key.

Acct

Accounting information

SClassId

Storage Class ID values as defined in the bit file server. The value will be mapped onto Storage Server space maps to locate free space of an appropriate type.

Form

Composition of media:

Striped

Concatenated

ParityStriped

Mirrored

Blksz

The block size in bytes of the media associated with this virtual volume.

StripeWidth

Number of stripes.

Flags

VV_WRITE_PROTECT VV may not be written.

VV_EOM VV has reached EOM (tape).

VV_OFFLINE VV is off-line (disk).

RefCnt

The number of links made to a the virtual_volume_md. When a link is deleted, the reference count is decremented, and when zero, the record becomes eligible for deletion.

NumReads

Number of read operations on this virtual volume.

NumWrites

Number of write operations on this virtual volume.

OperationalState

Operational state flags:

ST_ENABLED
ST_DISABLED
ST_BROKEN

UsageState

Usage state flags:

ST_IDLE
ST_ACTIVE
ST_BUSY
ST_UNKNOWN

AdministrativeState

Administrative state flags:

ST_LOCKED
ST_UNLOCKED
ST_REPAIRED

VVState

Virtual volume state flags.

| | |
|------------------------|---|
| VOL_NULL | Volume is invalid and not in use. |
| VOL_SCRATCH | Volume has not be assigned and contains no valid data. |
| VOL_ALLOCATED | The volume has been assigned to a client. |
| VOL_FOREIGN | The volume contains data that was not written by HPSS. The format of the media is unknown. |
| VOL_HPSS_IMPORT | The volume has been imported into HPSS from another site. The format is either HPSS, UniTree, or CFS. |
| VOL_ALLOCATED_AND_FULL | The volume is in HPSS format and has been written to the end. (Tape only). |

StripeLength

Rotation interval in bytes.

EstimatedSize

The estimated size of this virtual volume. For disk this is equal to the 'ActualSize' field.

ActualSize

The size of this virtual volume. On tape this is the number of bytes currently written. On disk this is the size

of the entire device.

NextByteAddr

The relative address of the next byte to write within this virtual volume (not used in disk Storage Servers).

NextAbsoluteAddr

Concatenated media absolute addresses that define the ending position of the last write operation on this volume (not used in tape Storage Servers).

Owners

Owner is an array of `hpsoid_t` that define the owners of this virtual volume. Each client wishing to be linked with this virtual volume provides a non-zero SOID that can be used by the Storage Server to contact the client. Storage maps associated with the virtual volume will be identified with a SOID. Zero entries in the array are vacant. Each time a non-zero entry is added to the array, `RefCnt` is incremented by one.

Security

Security information of unknown format.

LastRead

The time of last read.

LastWrite

The time of last write.

Creation

The time the virtual volume was created.

Update

The time of the last change to the virtual volume.

FamilyId

The family associated with the virtual volume. This field is zero for tapes not associated with families, and on all disks.

Clients

The following clients access the data definition:

Storage segment layer, Storage Map layer, Storage System Manager.

4.2.12. **Physical Volume Record - `physical_volume_record_t`**

Description

A physical volume record is a volatile record that describes a physical storage volume.

Format

The physical volume record has the following format:

```
typedef struct physical_volume_record {
    char                PVName[HPSS_PV_NAME_SIZE];
    unsigned32         FormatFlags;
    unsigned32         Flags;
    pthread_cond_t     Cond;
    signed32           ThreadsUsing;
    tran_tid_t         OwningTid;
}
```

```
tran_tid_t          CreateTid;
tran_tid_t          DeleteTid;
relative_address_t  CurrentRelativeAddr;
absolute_address_t  CurrentAbsoluteAddr;
relative_address_t  NextRelativeAddr;
absolute_address_t  NextAbsoluteAddr;
u_signed64          NumSectionBytes;
unsigned32          BlockSize;
unsigned32          BlocksBetweenTMs;
signed32            VolumeState;
signed32            AdministrativeState;
hpss_connect_handle_t ConnectionHandle;
hpss_object_handle_t SessionHandle;
device_table_record_t *DT;
physical_volume_md_t *MD;
struct physical_volume_record *SharedRecordPtr;
struct physical_volume_record *next_active, *prev_active;
struct physical_volume_record *next_sess, *prev_sess;
signed32            NumActiveSessions;
waitlist_t          *WaitListHead;
waitlist_t          *WaitListTail;
waitlist_t          *UnmountWaiter;
} physical_volume_record_t;
```

PVName

The identifier of the Physical Volume.

FormatFlags

PV recorded format. See pvl_definitions.idl for definitions.

Flags

| | |
|--------------------------------|---|
| PV_DIRTY | The metadata is dirty and should be flushed to SFS. |
| PV_INUSE | Cache entry in use. |
| PV_DESTROYED | Cache entry must be removed. |
| PV_BADMD | Cache MD must be reloaded once more. |
| PV_READ_WITHOUT_LOCK | Cache MD was last read without using a bracket read lock (possibly a dirty record). |
| PV_REREAD_ON_FINISH | Cache MD must be reloaded after transaction commits. |
| PV_DEALLOC_IN_PVL_ON_DESTROYPV | must be deallocated in the PVL when a PV destroy function commits. |

Cond

Condition structure to wait on for exclusive access.

ThreadsUsing

Count of the number of threads using or waiting for the cache entry.

OwningTid, CreateTid, DeleteTid

Transaction IDs that control use of the cache entry.

CurrentRelativeAddr

Append address of the current I/O (not used in disk Storage Servers). Only valid for append type media. This address is relative to the beginning of the media.

CurrentAbsoluteAddr

Append address of the current I/O (not used in disk Storage Servers). Only valid for append type media that support absolute address modes.

NextRelativeAddr

Append address of the last successfully complete append operation (not used in disk Storage Servers). Only valid on append type media. This address is relative to the beginning of the media.

NextAbsoluteAddr

Append address of the last successfully completed append operation (not used in disk Storage Servers). Only valid on append style media that supports absolute addressing modes.

NumSectionBytes

Number of bytes between media marks. (Only valid for tape style media).

BlockSize

The block size of the media (not used in disk Storage Servers).

BlocksBetweenTMs

The number of PV blocks between mandatory tape marks.

VolumeState

| | |
|------------------------|---|
| VOL_NULL | Volume is invalid and not in use. |
| VOL_SCRATCH | Volume has not be assigned and contains no valid data. |
| VOL_ALLOCATED | The volume has been assigned to a client. |
| VOL_FOREIGN | The volume contains data that was not written by HPSS. The format of the media is unknown. |
| VOL_HPSS_IMPORT | The volume has been imported into HPSS from another site. The format is either HPSS, UniTree, or CFS. |
| VOL_ALLOCATED_AND_FULL | The volume is in HPSS format and has been written to the end. (Tape only). |

AdministrativeState

Administrative state flags.

ST_LOCKED
ST_UNLOCKED

ConnectHandle

The connection on which the volume is currently mounted.

SessionHandle

The session on which the volume is currently mounted.

DT

The description of the device on which the media is mounted.

MD

Cached metadata information. This field is currently not used on tape. For disk if the pointer is NULL then there is no cached data, otherwise the data that is cached is valid.

SharedRecordPtr

Points to a shared `physical_volume_record` that contains mount information (disk only).

next_active, prev_active

Pointers to other structures that describe all the volumes currently mounted.

next_sess, prev_sess

Pointers to other structures that describe all the volumes currently mounted by this session.

NumActiveSessions

The number of sessions that desire access to this volume.

WaitListHead, WaitListTail

Pointers to structures that allow threads to wait for this volume to be dismounted.

UnmountWaiter

Pointer to a structure in which a thread which is unmounting this volume is waiting.

Clients

The following clients access the data definition:

Storage Server internals.

4.2.13. **Physical Volume Attribute Record - `pv_attr_t`**

Description

The `pv_attr_t` is used to transfer information about physical volumes across Storage Server functional interfaces.

Format

The `pv_attr_t` has the following format:

```
typedef struct pv_attr {
    physical_volume_md_t MD;
    u_signed64 RegisterBitMap;
    relative_address_t CurrentRelativeAddr;
    absolute_address_t CurrentAbsoluteAddr;
    device_table_record_t DeviceTable;
} pv_attr_t;
```

The following fields can be written during the process of creating a physical volume, but remain fixed afterwards:

| Name | Type | BlockSize |
|-------------|-------------|-----------|
| MinTMBlocks | MaxTMBlocks | |

The following fields can be modified with a `ss_PVSetAttributes` function call:

| | | |
|---------------------|----------------|---------------|
| VVID | Acct | EstimatedSize |
| CurrentRelativeAddr | LastMaint | InService |
| CurrentAbsoluteAddr | ActualSize | Security |
| VVSequence | RegisterBitMap | |

The remaining fields in the attribute record are readable but not writeable.

MD

The physical volume metadata record.

RegisterBitMap

The map of registered notification fields.

CurrentRelativeAddr

Current relative position of media (not used in disk Storage Servers). This field is an address relative to the beginning of the media and is only valid on sequential type media.

CurrentAbsoluteAddr

Current absolute position of media (not used in disk Storage Servers). This field is only valid on sequential type media that supports absolute addressing.

DeviceTable

When the physical volume is mounted, this field will contain information that describes the device on which the volume is mounted.

Clients

The following clients access the data definition:

Bitfile Server, Storage System Manager.

4.2.14. **Physical Volume Metadata - `physical_volume_md_t`**

Description

A `physical_volume_md` is a permanent record that describes a physical storage volume known to the Storage Server.

Format

The `physical_volume_md` has the following format:

```
typedef struct physical_volume_md {
    char                Name[PV_NAME_SIZE];
    media_type_t        Type;
    unsigned32          BlockSize
    unsigned32          MinTMBlocks
    unsigned32          MaxTMBlocks
    unsigned32          FormatFlags;
```

```
    unsigned32      Flags;
    signed32       OperationalState;
    signed32       UsageState;
    signed32       AdministrativeState;
    signed32       PVState;
    signed32       MountCntSinceService;
    signed32       MountCntSinceMaint;
    signed32       VVSequence
    signed32       RefCnt;
    acct_rec_t     Acct;
    u_signed64     EstimatedSize;
    u_signed64     ActualSize;
    relative_address_t NextWriteAddr;
    absolute_address_t NextAbsoluteWriteAddr
    timestamp_sec_t Creation;
    timestamp_sec_t Update;
    timestamp_sec_t LastRead;
    timestamp_sec_t LastWrite;
    timestamp_sec_t LastMaint;
    timestamp_sec_t InService;
    security_t     Security;
    hpssoid_t      VVID;
    unsigned32     FamilyId;
    unsigned32     Unused[7];
} physical_volume_md_t;
```

Name

This is the blank filled ASCII name of the physical volume and must be unique. It is the primary search key.

Type

Type of media:

NETWORK_ATTACHED_DISK

DIRECTLY_ATTACHED_DISK

BlockSize

The block size in bytes of the physical volume.

MinTMBlocks

The minimum allowable blocks between tape marks (not used in disk Storage Servers).

MaxTMBlocks

The maximum allowable blocks between tape marks (not used in disk Storage Servers).

FormatFlags

Flags that describe the recording format. See mvr_devdesc.idl.

Flags

PV_WRITE_PROTECT PV not writeable.

PV_EOM EOM has been reached on the volume.

PV_ABSADDR Absolute addressing enabled on the volume.

PV_OFFLINE PV is not mounted and not available (disk only).

OperationalState

Operational state flags:

ST_ENABLED

ST_DISABLED

ST_BROKEN

UsageState

Usage state flags:

ST_IDLE

ST_ACTIVE

ST_BUSY

AdministrativeState

Administrative state flags:

ST_LOCKED

ST_UNLOCKED

ST_REPAIRED

PVState

| | |
|------------------------|---|
| VOL_NULL | Volume is invalid and not in use. |
| VOL_SCRATCH | Volume has not be assigned and contains no valid data. |
| VOL_ALLOCATED | The volume has been assigned to a client. |
| VOL_FOREIGN | The volume contains data that was not written by HPSS. The format of the media is unknown. |
| VOL_HPSS_IMPORT | The volume has been imported into HPSS from another site. The format is either HPSS, UniTree, or CFS. |
| VOL_ALLOCATED_AND_FULL | The volume is in HPSS format and has been written to the end. (Not valid on disk). |

MountCntSinceService

The number of mounts performed on this physical volume since it was serviced.

MountCntSinceMaint

The number of mounts performed on this physical volume since it was serviced.

VVSequence

The sequence number of the physical volume in the virtual volume.

RefCnt

The number of links currently active for this physical volume.

Acct

Unknown accounting information.

EstimatedSize

The estimated length in bytes which remain on this physical volume. Used for tape only. For disk this field will be set equal to the 'ActualSize' field.

ActualSize

If tape, the number of bytes currently written to the tape. If disk the actual size of the device in bytes. This must be a multiple of the block size.

NextWriteAddr

Relative address of the next byte to be written on tape (not used in disk Storage Servers). The field is only valid for append style media. The address is relative to the beginning of the media.

NextAbsoluteWriteAddr

Absolute address of the next byte to be written to tape (not used in disk Storage Servers). This field is only valid for append style media. The address is only valid on media that supports absolute addressing modes.

Creation

The time the physical volume was created.

Update

The time of the last change to the physical volume.

LastRead

The time of last read.

LastWrite

The time of last write.

LastMaint

The time of last maintenance (the time the tape was last mounted and read without error).

InService

The time this volume was put into service.

Security

Security information of unknown format.

VVID

The Virtual Volume ID of which this physical volume is a member. This is a secondary key.

FamilyId

The family associated with the physical volume. This field is zero for tapes not associated with families, and on all disks.

4.2.15. **Device Table Record - device_table_record_t**

Description

A device table record is a volatile record that provides information about a storage device associated with a mounted physical volume.

Format

The `device_table_record` has the following format:

```
typedef struct device_table_record {
    signed32    MountId;
    signed32    DeviceID;
    signed32    Type;
    signed32    MvrFd;
    uuid_t      MvrId;
    char        MvrIP[MVR_IP_SIZE];
    char        PVName[HPSS_PV_NAME_SIZE];
} device_table_record_t;
```

MountId

The job identifier returned by the PVL that describes the mount of the volume.

DeviceID

Identification of the device on which the volume has been mounted.

MvrFd

A socket descriptor for sending control to the Mover.

MvrId

The `uuid_t` associated with the mover.

MvrIP

The mover's IP address, including port.

PVName

The ASCII name of the physical volume mounted.

4.2.16. Session Record - `ss_session_t`**Description**

The `ss_session_t` describes a Storage Server session in effect between the Storage Server and a client. Storage segments, virtual volumes, physical volumes, connections and sessions are interconnected in this structure so that if connection with a client fails, the appropriate recovery procedures can be carried out (unmount media, terminate writes in progress).

Format

The `ss_session_t` has the following format:

```
typedef struct ss_session {
    hpss_object_handle_t    SessionH;
    hpss_object_handle_t    ConnectH;
    unsigned32              Flags;
    signed32                 ReqstID;
    hpss_object_handle_t    *UniqueAddr;
    pthread_mutex_t         Lock;
    pthread_cond_t          WaitUnlock;
    pthread_cond_t          WaitVV;
    signed32                 ThreadsUsing;
```

```
    unsigned32                WantedFamily;
    unsigned32                WantedStorageClass;
    hpsoid_t                  AvoidVVID;
    timestamp_sec_t           TimeStamp;
    struct ss_session          *next_conn, *prev_conn;
    struct ss_session          *next_hash, *prev_hash;
    storage_segment_record_t  *HeadSegList, *TailSegList;
    virtual_volume_record_t   *HeadVVList, *TailVVList;
    physical_volume_record_t  *HeadPVList, *TailPVList;
    hpsoid_t                  CurrentVV;
} ss_session_t;
```

SessionH

Identifies the session object.

ConnectH

Identifies parent connection for this session.

Flags

| | |
|--------------------------------------|--|
| BUSY_SESSION | Session is locked and busy. |
| WAIT_FOR_SESSION | Busy session should be waited for. |
| SESSION_WAITING_FOR_VV | Session is waiting for a VV assignment. |
| FORCE_DISMOUNT | Force immediate tape dismounts. |
| SESSION_STORAGE_CLASS_OVERSUBSCRIBED | Session blocked by storage class oversubscription. |

ReqstID

The request ID performing an action on the session.

UniqueAddr

Pointer to a record on the stack, so that lower level calls can tell whether they were called directly by an RPC or by a straight function call.

Lock

Access lock for this structure.

WaitUnlock

Condition variable to wait on when waiting for another thread to unlock the session.

WaitVV

Condition variable to wait on when the session is waiting for a VV assignment.

ThreadsUsing:

Number of threads that have locked the record, or are waiting to lock the record.

WantedFamily

The needed family id of a tape the session is waiting for.

WantedStorageClass

The needed storage class of a tape the session is waiting for.

next_conn, prev_conn;

List of ss_sessions that are part a connection.

next_hash, prev_hash

List of all active sessions.

HeadSegList, TailSegList

List of storage segments which hold resources which must be recovered if a connection drops.

HeadVVLList, TailVVLList

List of virtual volumes involved in connections which must be recovered if a connection drops.

HeadPVLList, TailPVLList

List of physical volumes involved in connections which must be recovered if a connection drops.

4.2.17. Relative Address - relative_address_t**Description**

The relative address defines the address of a byte relative to the beginning of a set of media.

Format

The relative_address_t has the following format:

```
typedef struct relative_address {
    byte      Version;
    byte      Reserved;
    unsigned16 Partition;
    unsigned32 Section;
    u_signed64 Offset;
} relative_address_t;
```

Version

The format version number of this record.

Partition

The partition number that contains the byte addressed. (Set to zero for disk).

Section

The section number that contains the byte addressed. (Set to zero for disk).

Offset

The byte offset within the section. On tape, the byte offset from the last tape mark. On disk, the offset from the beginning of the device.

4.2.18. Composite Address - composite_address_t**Description**

The composite address defines a set of locations on a set of physical media. The individual locations are defined by media absolute addresses.

Format

The `composite_address_t` has the following format:

```
typedef struct composite_address {
    absolute_address_t    CompAddr[COMP_ADDR_SIZE];
} composite_address_t;
```

CompAddr

The array of absolute addresses.

4.2.19. Absolute Address - `absolute_address_t`

Description

The absolute address defines a location on a physical volume.

Format

The `absolute_address_t` has the following format:

```
typedef struct absolute_address {
    byte    AbsAddr[ABS_ADDR_SIZE];
} absolute_address_t;
```

AbsAddr

The device dependent physical volume address.

4.2.20. Physical Volume List - `pv_list_t`

Description

This data structure is used to pass a list of `pv_list_element_t`'s across the RPC interface.

Format

The `pv_list_t` has the following format:

```
typedef struct pv_list {
    signed32    Length
    [size_is(Length)] pv_list_element_t    List[*];
} pv_list_t;
```

Length

The number of elements in List.

List

A conformant array of `pv_list_element_t`'s that contain information about a number of Pvs.

Note

The order of elements in this list is determined by the order of physical volumes in a virtual volume. Length is the number of physical volumes in the virtual volume.

4.2.21. Physical Volume List Element - `pv_list_element_t`

Description

This is a general data structure that is used to make up lists of physical volumes.

Format

The `pv_list_t` has the following format:

```
typedef struct pv_list {
    char          Name[PV_NAME_SIZE];
    u_signed64    Flags;
} pv_list_element_t;
```

4.2.22. Owner Record - owner_rec_t**Description**

This data structure describes an element of a list of owners of a storage object.

Format

The `owner_rec_t` has the following format:

```
#define OWNER_SIZE 1
typedef struct owner_rec {
    u_signed64  Flags;
    hpsoid_t    OwnerArray[OWNER_SIZE];
} owner_rec_t;
```

OwnerArray

Array of SOIDs that identify an owner of the object associated with this record. Any or all of the array elements may be null.

Flags

`DELETE_OWNER` If set during a `SetAttributes` call, the owner will be deleted from the object's owner list, otherwise the owner is added.

Clients

The following clients access the data definition:

Bitfile Server, Storage System Manager.

4.2.23. Wait List - waitlist_t**Description**

The waitlist structure is used to queue threads that are waiting for resources. This structure is used mainly to queue threads that are waiting for media mounts.

Format

The waitlist structure has the following format:

```
typedef struct waitlist {
    pthread_cond_t  WaitCond;
    signed32        WaitFlag;
    signed32        Error;
    signed32        LayerDefined1;
    signed32        LayerDefined2;
    struct waitlist *Next;
} waitlist_t;
```

WaitCond

The condition variable on which to wait.

WaitFlag

The flag on which to wait.

Error

Error that occurred during the wait. If set the resource that the thread is waiting for can not be accessed at this time.

LayerDefined1, LayerDefined2

Currently unused.

Next

Pointer to the next element waiting for the resource.

4.2.24. **Storage Class Array - `ss_class_array_t`**

Description

This data structure is used to pass an array of `ss_class_t`'s across the RPC interface.

Format

The `ss_class_array_t` has the following format:

```
typedef struct ss_class_array {
    signed32          Length
    [size_is(Length)] ss_sclass_t  Array[*];
} ss_class_array_t;
```

Length

The number of elements in Array.

List

A conformant array of `ss_class_t` elements that contain information about the storage classes provided in the server.

4.2.25. **Storage Class Array Element - `ss_class_t`**

Description

This data structure is an element of a list of storage class statistics.

Format

The `ss_class_t` has the following format:

```
typedef struct ss_class {
    unsigned32      SClassId;
    u_signed64      TotalSpace;
    u_signed64      FreeSpace;
    struct ss_class *Next;
} ss_class_t;
```

SClassId

The storage class reported

TotalSpace

The amount of storage in the storage class. On tape, this is the number of virtual volumes in the storage class. On disk, this is the amount of storage in bytes.

FreeSpace

The amount of free storage in the storage class. On tape, this is the number of virtual volumes that have never been written. On disk, this is the amount of free storage, in bytes.

Next

Pointer to next list element, or null.

4.2.26. **Event Array - `ss_sclass_array_t`****Description**

This data structure is used to pass an array of `ss_event_rec_t`'s across the RPC interface.

Format

The `ss_event_rec_t` has the following format:

```
typedef struct ss_sclass_array {
    signed32                Length
    [size_is(Length)] ss_sclass_t    Array[*];
} ss_sclass_array_t;
```

Length

The number of elements in Array.

List

A conformant array of `ss_event_rec_t` elements that contain information about events.

4.2.27. **Event Array Element - `ss_event_rec_t`****Description**

This data structure is an element of an array of Storage Server events.

Format

The `ss_event_rec_t` has the following format:

```
typedef struct ss_event_rec {
    signed32                ReqstID;
    char                    RoutineName[K_EVENT_STRINGS];
    char                    EventName[K_EVENT_STRINGS];
    unsigned32              SerialNumber;
    unsigned32              Flags;
    unsigned32              tv_sec;
    unsigned32              tv_usec;
} ss_event_rec_t;
```

ReqstID

The request ID the event is recorded under.

RoutineName

The ASCII name of the function that recorded the event.

EventName

An ASCII name for the event.

SerialNumber

A serial number for the event, assigned by the event manager.

Flags

Event Flags - none defined.

tv_sec

Time in seconds when the event was recorded. From struct timeval.

4.2.28. Segment Array

Description

This data structure used to pass an array of storage segment SOIDs across the RPC interface.

Format

The `ss_segment_array_t` has the following format:

```
typedef struct ss_segment_array {
    signed32          Length
    [size_is(Length)] hpsoid_t  Array[*];
} ss_segment_array_t;
```

Length

The number of elements in Array

Array

A conformant array of storage segment SOIDs.

4.2.29. Delete segment array

Description

This data structure used to pass an array of storage segment SOIDs across the RPC interface, and return a list of error codes.

Format

The `ss_delete_segment_array_t` has the following format:

```
typedef struct ss_delete_segment_array {
    signed32          Length
    [size_is(Length)] ss_delete_segment_t  Array[*];
} ss_delete_segment_array_t;
```

Length

The number of elements in Array

Array

A conformant array of storage segment delete elements

4.2.30. Delete Segment Array Element

Description

This data structure describes a storage segment to be deleted and the delete error code.

Format

The `ss_delete_segment_t` has the following format:

```
typedef struct ss_delete_segment {
    hpsoid_t  SSID;
    signed32  *Error;
} ss_delete_segment_t;
```

SSID

The SOID of the storage segment to delete.

Error

A pointer to the error code that resulted from the segment deletion.

4.2.31. Copy Control Block

Description

The copy control block contains all of the information needed to copy or move a storage segment to another segment or location.

Format

The `copy_control_block_t` record has the following format:

```
typedef struct copy_control_block {
    tran_tid_t          Tid;
    pthread_mutex_t     Lock;
    pthread_cond_t      Cond;
    unsigned32          Flags;
    ss_iolist_t         *ReadSeg, *WriteSeg;
    IOD_t               ReadIOD, WriteIOD;
    srcsinkdesc_t       ReadSrcSSD, WriteSrcSSD,
                       WriteSinkSSD;
    requestspec_t       WriteRequestSpec,
                       ReadRequestSpec;
    u_signed64          ReadBlockSize;
    unsigned32          ReadStripeWidth;
    signed32            ReadError, WriteError;
    IOR_t               ReadIOR, WriteIOR;
    srcsinkreply_t      ReadSrcSSR, ReadSinkSSR,
                       WriteSrcSSR, WriteSinkSSR;
    signed32            ReqstID;
    trpc_handle_t       Binding;
    hpss_object_handle_t *ActiveSessionHandle,
                       *PassiveSessionHandle;
    signed32            SockAddr,
                       SockPort,
                       SockFamily;
    u_signed64          TransferID;
} copy_control_block_t;
```

Tid

The transaction to use during the operation.

Lock

Synchronizes use of the record.

Cond

Synchronizes use of the record.

Flags

| | |
|------------------|-------------------------------|
| CB_ABORT | Copy operation aborted |
| CB_ADDR_LOADED | Addresses have been loaded |
| CB_READ_COMPLETE | Read step of copy is complete |

ReadSeg, WriteSeg

Information about the segments to be read and written.

ReadIOD, WriteIOD

The IODs to use in copying the segment.

ReadSrcSSD, WriteSrcSSD, WriteSinkSSD

The read and write IOD source and sink descriptors.

WriteRequestSpec, ReadRequestSpec

The request specific fields used in the copy operation.

ReadBlockSize

The VV blocksize of the segment to be copied.

ReadStripeWidth

The VV stripe width of the segment to be copied.

ReadError, WriteError

The error codes returned from the segment read and write operations.

ReadIOR, WriteIOR

The IOR structures which return the results of the segment read and write operations.

ReadSrcSSR, ReadSinkSSR, WriteSrcSSR, WriteSinkSSR

The source and sink reply components of ReadIOR and WriteIOR.

ReqstID

The request ID used in the copy operation.

Binding

The binding used in the copy operation.

ActiveSessionHandle

The session used on the active side of the copy.

PassiveSessionHandle

The session used on the passive side of the copy.

SockAddr, SockPort, SockFamily

The information that describes the socket used to connect to the mover performing the copy.

TransferID

The transfer ID assigned by the mover for the copy operation.

5. Mover Functions

This chapter specifies the Mover programming interface. Specifically, the following information is provided:

Application Programming Interfaces (APIs)

Data Definitions

5.1. API Functions

This section describes all APIs which are provided for use by another HPSS subsystem or by a client external to HPSS. The API interface specification includes the following information:

Name

Syntax

Description

Parameters

Return Values

Error Conditions

Related Information

Clients

Notes

5.1.1. `mvr_Abort`

Purpose

Abort the current request outstanding on the current Mover connection.

Syntax

```
#include "hpss_iod.h"

void
mvr_Abort(
    IOD_t *IOD);    /* IN */
```

Description

The `mvr_Abort` function is called to abort the current request. If the *Count* field in the request specific information structure is equal to zero, the Mover aborts the request immediately; otherwise the request is aborted after at least the number of bytes specified by the *Count* field have been successfully transferred.

Parameters

| | |
|--------------------------------------|---|
| <i>IOD->RequestID</i> | Request identifier. |
| <i>IOD->Function</i> | Set to IOD_ABORT . |
| <i>IOD->ReqSpecInfo->Count</i> | If zero, immediately abort the request, abort request after transferring at least this number of bytes. |

Return values

None. Note that this request will not generate an IOR.

Error conditions

None. Note that if an abort request is sent after the data movement request has completed, the abort request is ignored.

See also

`mvr_Read`, `mvr_Write`.

Clients

Storage Server.

Notes

None.

5.1.2. mvr_CreateDevice

Purpose

Add a new device.

Syntax

```
#include "mvr_dceif.h"
```

```
signed32
```

```
mvr_CreateDevice(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    signed32               RequestID,       /* IN */
    device_desc_t         *DeviceMDPtr,     /* IN */
    trpc_status_t         *RPCError);       /* OUT */
```

Description

The **mvr_CreateDevice** routine is called to create a new entry in the Mover device table (both for the current execution of the Mover and in Mover device metadata).

Parameters

| | |
|--------------------|--|
| <i>Binding</i> | The TRPC binding handle. |
| <i>CNH</i> | Pointer to an HPSS connection handle. |
| <i>RequestID</i> | Request ID for this query. |
| <i>DeviceMDPtr</i> | Pointer to a <code>device_desc_t</code> structure that contains the metadata for the new device. |
| <i>RPCError</i> | Pointer to status location; will contain RPC error indication. |

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|----------------|--|
| HPSS_EAGAIN | The Mover is currently in the process of shutting down. |
| HPSS_EBADCONN | Connection handle is not valid for this request. |
| HPSS_EEXIST | The specified device already exists. |
| HPSS_EINVAL | Invalid attribute value. |
| HPSS_EMMINSERT | An error occurred while attempting to write the new device metadata. |
| HPSS_EOFDGET | An error occurred while trying to open the Mover device metadata file. |
| HPSS_EPERM | Client not authorized for this request. |

See also

mvr_DeleteDevice.

Clients

Storage System Management.

Notes

None.

5.1.3. mvr_DeleteDevice

Purpose

Remove an existing device.

Syntax

```
#include "mvr_dceif.h"

signed32
mvr_DeleteDevice(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,           /* IN */
    signed32               RequestID,       /* IN */
    unsigned32             DeviceID,        /* IN */
    trpc_status_t          *RPCError);      /* OUT */
```

Description

The `mvr_DeleteDevice` routine is called to remove an existing entry from the Mover device table (both for the current execution of the Mover and in Mover device metadata).

Parameters

| | |
|------------------|--|
| <i>Binding</i> | The TRPC binding handle. |
| <i>CNH</i> | Pointer to an HPSS connection handle. |
| <i>RequestID</i> | Request ID for this query. |
| <i>DeviceID</i> | The identifier of the device to be deleted. |
| <i>RPCError</i> | Pointer to status location; will contain RPC error indication. |

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|----------------|--|
| HPSS_EAGAIN | The Mover is currently in the process of shutting down. |
| HPSS_EBADCONN | Connection handle is not valid for this request. |
| HPSS_EBADF | The specified device does not exist or is not configured for this Mover. |
| HPSS_EINVAL | Invalid attribute value. |
| HPSS_EMMDELETE | An error occurred while attempting to delete the device metadata. |
| HPSS_EOFDGET | An error occurred while trying to open the Mover device metadata file. |
| HPSS_EPERM | Client not authorized for this request. |

See also

`mvr_CreateDevice`.

Clients

Storage System Management.

Notes

None.

5.1.4. mvr_DeviceGetAttrs

Purpose

Query current attribute values for a device controlled by the Mover.

Syntax

```
#include "mvr_dceif.h"
```

```
signed32
```

```
mvr_DeviceGetAttrs(
    trpc_handle_t           Binding,           /* IN */
    hpss_connect_handle_t *CNH,             /* IN */
    signed32               RequestID,        /* IN */
    unsigned32             DeviceID,         /* IN */
    devdesc_attr_t        *DevAttrOut,      /* OUT */
    trpc_status_t          *RPCError);       /* OUT */
```

Description

The **mvr_DeviceGetAttrs** function is called to query attribute values for a device controlled by the Mover.

Parameters

| | |
|-------------------|--|
| <i>Binding</i> | The TRPC binding handle. |
| <i>CNH</i> | Pointer to an HPSS connection handle. |
| <i>RequestID</i> | Request ID for this query. |
| <i>DeviceID</i> | Identifier of device being queried. |
| <i>DevAttrOut</i> | Pointer to a devdesc_attr_t structure that will contain the results of the query. |
| <i>RPCError</i> | Pointer to status location; will contain RPC error indication. |

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|---------------|---|
| HPSS_EAGAIN | The Mover is currently in the process of shutting down. |
| HPSS_EBADF | No such device. |
| HPSS_EBADCONN | Connection handle is not valid for this request. |
| HPSS_EPERM | Client not authorized for this request. |

See also

mvr_DeviceSetAttrs, **mvr_DeviceGetAttrs_IOD**, **mvr_DeviceSetAttrs_IOD**.

Clients

Storage System Management.

Notes

None.

5.1.5. mvr_DeviceGetAttrs_IOD

Purpose

Query the current setting of attributes for a device controlled by the Mover.

Syntax

```
#include "hpss_iod.h"

signed32
mvr_DeviceGetAttrs_IOD(
    IOD_t *IOD,      /* IN */
    IOR_t *IOR);    /* OUT */
```

Description

The **mvr_DeviceGetAttrs_IOD** function is called to query the current attributes of a device that is controlled by the Mover.

Parameters

| | |
|--|-----------------------------------|
| <i>IOD->RequestID</i> | Request identifier. |
| <i>IOD->Function</i> | Set to IOD_GETDEVICEATTR . |
| <i>IOD->ReqSpecInfo.DeviceID</i> | Device identifier. |
| <i>IOR->RequestID</i> | Request identifier. |
| <i>IOR->Status</i> | Status of request. |
| <i>IOR->ReqSpecReply.ReqReplyType</i> | Set to REPLY_DEVICEATTR.. |
| <i>IOR->ReqSpecReply.ReqReply_u</i> | Returned device attributes. |

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|-------------|---|
| HPSS_EBADF | No such device. |
| HPSS_ERANGE | Device ID specifies the generic Mover device. |

See also

mvr_DeviceSetAttrs_IOD, **mvr_DeviceGetAttrs**, **mvr_DeviceSetAttrs**.

Clients

Storage Server.

Notes

None.

5.1.6. mvr_DeviceSetAttrs

Purpose

Alter current attribute values for a device controlled by the Mover.

Syntax

```
#include "mvr_dceif.h"
```

```
signed32
```

```
mvr_DeviceSetAttrs(  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNH,            /* IN */  
    signed32               RequestID,       /* IN */  
    u_signed64             InSelectBitmap,   /* IN */  
    u_signed64             OutSelectBitmap,  /* OUT */  
    devdesc_attr_t        DevAttrIn,       /* IN */  
    devdesc_attr_t        DevAttrOut,      /* OUT */  
    trpc_status_t         *RPCError);      /* OUT */
```

Description

The **mvr_DeviceSetAttrs** function is called to alter attribute values for a device controlled by the Mover. This interface should NOT be used to change the current read/write position of a device (see **mvr_DeviceSetAttrs_IOD**). The attribute which may be changed using this routine are:

- *SecurityLabel*
- *RegisterBitmap*
- *Flags*
- *NumberOfErrors* (reset only)
- *BytesRead* (reset only)
- *BytesWritten* (reset only)
- *AdministrativeState* (**ST_REPAIRED**, **ST_LOCKED**, **ST_UNLOCKED**)

Parameters

| | |
|------------------------|--|
| <i>Binding</i> | The TRPC binding handle. |
| <i>CNH</i> | Pointer to an HPSS connection handle. |
| <i>RequestID</i> | Request ID for this query. |
| <i>InSelectBitmap</i> | Indicates which device attributes are to be changed. |
| <i>OutSelectBitmap</i> | Indicated which device attributes were changed as a result of this request. |
| <i>DevAttrIn</i> | Pointer to a devdesc_attr_t structure that contains the desired new attribute values. |

| | |
|-------------------|--|
| <i>DevAttrOut</i> | Pointer to a devdesc_attr_t structure that will contain the results of the attribute changes. |
| <i>RPCError</i> | Pointer to status location; will contain RPC error indication. |

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|---------------|---|
| HPSS_EAGAIN | The Mover is currently in the process of shutting down. |
| HPSS_EBADF | No such device. |
| HPSS_EBUSY | Device is currently in use. |
| HPSS_EINVAL | Invalid attribute value. |
| HPSS_EBADCONN | Connection handle is not valid for this request. |
| HPSS_EPERM | Client not authorized for this request. |

See also

mvr_DeviceGetAttrs, mvr_DeviceGetAttrs_IOD, mvr_DeviceSetAttrs_IOD.

Clients

Storage System Management.

Notes

None.

5.1.7. mvr_DeviceSetAttrs_IOD

Purpose

Alter current attribute values for a device controlled by the Mover.

Syntax

```
#include "hpss_iod.h"

signed32
mvr_DeviceSetAttrs_IOD(
    IOD_t *IOD,      /* IN */
    IOR_t *IOR);    /* OUT */
```

Description

The **mvr_DeviceSetAttrs_IOD** function is called to change attribute values for a device controlled by the Mover. This interface should be used when changing the current read/write position of a device.

Parameters

| | |
|--|--|
| <i>IOD->RequestID</i> | Request identifier. |
| <i>IOD->Function</i> | Set to IOD_SETDEVICEATTR . |
| <i>IOD->ReqSpecInfo.DeviceID</i> | Device identifier. |
| <i>IOD->ReqSpecInfo.Type</i> | Set to INFO_DEVICEATTR . |
| <i>IOD->ReqSpecInfo.ReqInfo_u</i> | New device attributes. |
| <i>IOR->RequestID</i> | Request identifier. |
| <i>IOR->Status</i> | Status of request. |
| <i>IOR->ReqSpecReply.ReqReplyType</i> | Set to REPLY_DEVICEATTR . |
| <i>IOR->ReqSpecReply.ReqReply_u</i> | Returned device attributes, after completion of operation. |

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|-----------------|---|
| HPSS_EBADF | No such device. |
| HPSS_EBUSY | The device is busy satisfying another request. |
| HPSS_EINVAL | Invalid attribute value(s). |
| HPSS_EOPNOTSUPP | Attempt to set position when volume flags specifies an unsupported media format type. |
| HPSS_EPERM | Attempt to change device state that would be inconsistent with the device metadata. |

HPSS_ERANGE

Device ID specifies the generic Mover device.

See also

`mvr_DeviceGetAttrs_IOD`, `mvr_DeviceGetAttrs`, `mvr_DeviceSetAttrs`.

Clients

Storage Server.

Notes

None.

5.1.8. mvr_DeviceSpec

Purpose

Performs device specific operation.

Syntax

```
#include "hpss_iod.h"

signed32
mvr_DeviceSpec(
    IOD_t *IOD,          /* IN */
    IOR_t *IOR);        /* OUT */
```

Description

The **mvr_DeviceSpec** function is called to perform device specific operations.

Parameters

| | |
|---|--|
| <i>IOD->RequestID</i> | Request identifier. |
| <i>IOD->Function</i> | Set to IOD_DEVICESPEC . |
| <i>IOD->ReqSpecInfo.Flags</i> | Processing options. Valid values include HOLD_RESOURCES and NO_LABEL_CHECK. |
| <i>IOD->ReqSpecInfo.DeviceID</i> | Device identifier. |
| <i>IOD->ReqSpecInfo.SubFunctionSet</i> | Set depending on function desired: DEVICE_LOAD DEVICE_UNLOAD DEVICE_FLUSH DEVICE_WRITETM (tape only) DEVICE_LOADDISPLAY DEVICE_READLABEL DEVICE_WRITELABEL DEVICE_CLEAR (disk only) |
| <i>IOD->ReqSpecInfo.InfoType</i> | Set to reflect information type, if any: INFO_LOADDISPLAY - for load display request INFO_VOLUMEID - for write label, unload, flush, write tape mark, clear, unload requests INFO_NONE - for read label, unload (if label check bypassed) |
| <i>IOD->ReqSpecInfo.ReqInfo_u.</i> | Command specific information. |
| <i>IOR->RequestID</i> | Request identifier. |
| <i>IOR->Status</i> | Status of request. |
| <i>IOR->ReqSpecReply.ReqReplyType</i> | Set to indicate returned information, if any: REPLY_VOLUMEID - for read label requests REPLY_NONE - other requests. |

IOR->ReqSpecReply.ReqReply_u Returned command specific information.

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|-----------------|--|
| HPSS_EBADF | No such device. |
| HPSS_EBUSY | Device is currently in use by another request. |
| HPSS_EINVAL | Invalid operation information. |
| HPSS_EIO | An I/O error occurred. |
| HPSS_EWRPROTECT | Media is read-only. |

See also

None.

Clients

Storage Server, Physical Volume Library.

Notes

None.

5.1.9. mvr_MVRGetAttrs

Purpose

Query current state of the Mover.

Syntax

```
#include "mvr_dceif.h"
```

```
signed32
```

```
mvr_MVRGetAttrs(  
    trpc_handle_t          Binding,      /* IN */  
    hpss_connect_handle_t *CNH,        /* IN */  
    signed32               RequestID,   /* IN */  
    mover_attr_t          *MvrAttrOut,  /* OUT */  
    trpc_status_t         *RPCError);   /* OUT */
```

Description

The **mvr_MVRGetAttrs** function is called to query attribute values for the state of the Mover.

Parameters

| | |
|-------------------|--|
| <i>Binding</i> | The TRPC binding handle. |
| <i>CNH</i> | Pointer to an HPSS connection handle. |
| <i>RequestID</i> | Request ID for this query. |
| <i>MvrAttrOut</i> | Pointer to a mover_attr_t structure to hold the results of the query. |
| <i>RPCError</i> | Pointer to status location; will contain RPC error indication. |

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|---------------|--|
| HPSS_EBADCONN | Connection handle is not valid for this request. |
| HPSS_EPERM | Client not authorized for this request. |

See also

mvr_MVRSetAttrs.

Clients

Storage System Management.

Notes

None.

5.1.10. `mvr_MVRSetAttrs`**Purpose**

Alter the current state attributes of the Mover.

Syntax

```
#include "mvr_dceif.h"
```

```
unsigned32
```

```
mvr_MVRSetAttrs(
    trpc_handle_t          Binding,          /* IN */
    hpss_connect_handle_t *CNH,            /* IN */
    signed32               RequestID,       /* IN */
    u_signed64             InSelectBitmap,   /* IN */
    u_signed64             *OutSelectBitmap, /* OUT */
    mover_attr_t           *MvrAttrIn,      /* IN */
    mover_attr_t           *MvrAttrOut,     /* OUT */
    trpc_status_t         *RPCError);       /* OUT */
```

Description

The `mvr_MVRSetAttrs` function is called to alter attribute values for the current state of the Mover. The attributes which may be set are:

- *NumberOfRequestsProcessed* (reset only).
- *NumberOfDataTransfers* (reset only).
- *NumberOfRequestErrors* (reset only).
- *NumberOfBytesMoved* (reset only).
- *TotalMoveTime* (reset only).
- *RegisterBitmap*.
- *BufferSize*.

Parameters

| | |
|------------------------|--|
| <i>Binding</i> | The TRPC binding handle. |
| <i>CNH</i> | Pointer to an HPSS connection handle. |
| <i>RequestID</i> | Request ID for this query. |
| <i>InSelectBitmap</i> | Indicates which Mover state attributes are to be changed. |
| <i>OutSelectBitmap</i> | Indicated which Mover state attributes were changed as a result of this request. |
| <i>MvrAttrIn</i> | Pointer to a <code>mover_attr_t</code> structure that contains the desired new Mover state attribute values. |
| <i>MvrAttrOut</i> | Pointer to a <code>mover_attr_t</code> structure to hold the attribute values after the operation. |

RPCError Pointer to status location; will contain RPC error indication.

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

HPSS_EAGAIN The Mover is currently in the process of shutting down.

HPSS_EINVAL Invalid attribute value.

HPSS_EBADCONN Connection handle is not valid for this request.

HPSS_EPERM Client not authorized for this request.

See also

mvr_MVRGetAttrs.

Clients

Storage System Management.

Notes

None.

5.1.11. mvr_Read

Purpose

Read data from a device or devices controlled by the Mover.

Syntax

```
#include "hpss_iod.h"

signed32
mvr_Read(
    IOD_t *IOD,          /* IN */
    IOR_t *IOR);        /* OUT */
```

Description

The **mvr_Read** function is called to read data from a device controlled by the Mover and send that data to another Mover (HPSS or client Mover). The description of the request is contained in the structure pointed to by the IOD parameter and results of the read are returned in the structure pointed to by the IOR parameter.

Parameters

| | |
|--------------------------------|--|
| <i>IOD->RequestID</i> | Request identifier. |
| <i>IOD->Function</i> | Set to IOD_READ . |
| <i>IOD->Flags</i> | Processing options (see IOD/IOR design specification for details). Valid values include REPLYWHENREADY and LAST_IN_XFER. |
| <i>IOD->SrcDescLength</i> | Number of source descriptors. |
| <i>IOD->SinkDeskLength</i> | Number of sink descriptors. |
| <i>IOD->SrcDescList</i> | List of descriptors describing data source. |
| <i>IOD->SinkDescList</i> | List of descriptors describing data sink. |
| <i>IOR->RequestID</i> | Request identifier. |
| <i>IOR->Flags</i> | IOR_COMPLETE indicates request is complete. IOR_ERROR indicates an error was encountered. |
| <i>IOR->Status</i> | Status of the request. |
| <i>IOR->SrcReplyLength</i> | Number of source reply descriptors. |
| <i>IOR->SinkReplyLength</i> | Number of sink reply descriptors. |
| <i>IOR->SrcReplyList</i> | List of descriptors describing data source status. |
| <i>IOR->SinkReplyList</i> | List of descriptors describing data sink status. |

Return values

A value of zero (0) is returned upon successful completion of the read request. If an error is detected, a value is returned that indicates the specific error condition, possible errors are listed below.

Error conditions

| | |
|-----------------|--|
| HPSS_EABORT | Request was aborted. |
| HPSS_EBADF | Invalid device identifier. |
| HPSS_EINVAL | An offset or resulting offset would exceed the capability of the device. |
| HPSS_EIO | An I/O error occurred. |
| HPSS_EOPNOTSUPP | Requested operation is not supported (e.g., IPI-3 data transfer request not supported by Mover). |
| HPSS_ERANGE | A value specified in the IOD is out of range. |
| HPSS_ESYSTEM | Operating system service failed. |
| HPSS_ETRANSFER | Transfer of data to client failed. |

See also

mvr_Write.

Clients

Storage Server.

Notes

None.

5.1.12. `mvr_ServerGetAttrs`**Purpose**

Query current state of the Mover's server object.

Syntax

```
#include "mvr_dceif.h"
```

```
signed32
```

```
mvr_ServerGetAttrs(
    trpc_handle_t           Binding,           /* IN */
    hpss_connect_handle_t  *CNH,             /* IN */
    signed32               RequestID,        /* IN */
    hpss_server_attr_t     *SvrAttrOut,      /* OUT */
    trpc_status_t          *RPCError);      /* OUT */
```

Description

The `mvr_ServerGetAttrs` function is called to query attribute values for the state of the Mover's server object.

Parameters

| | |
|-------------------|---|
| <i>Binding</i> | The TRPC binding handle. |
| <i>CNH</i> | Pointer to an HPSS connection handle. |
| <i>RequestID</i> | Request ID for this query. |
| <i>SvrAttrOut</i> | Pointer to a <code>server_attr_t</code> structure to hold the results of the query. |
| <i>RPCError</i> | Pointer to status location; will contain RPC error indication. |

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|---------------|--|
| HPSS_EBADCONN | Connection handle is not valid for this request. |
| HPSS_EPERM | Client not authorized for this request. |

See also

`mvr_ServerSetAttrs`.

Clients

Storage System Management.

Notes

The `hpss_server_attr_t` structure is the common HPSS server object attribute structure.

5.1.13. mvr_ServerSetAttrs

Purpose

Alter the current state attributes of the Mover's server object.

Syntax

```
#include "mvr_dceif.h"
```

```
signed32
```

```
mvr_ServerSetAttrs(  
    trpc_handle_t          Binding,          /* IN */  
    hpss_connect_handle_t *CNH,           /* IN */  
    signed32              RequestID,       /* IN */  
    u_signed64            InSelectBitmap,   /* IN */  
    u_signed64            OutSelectBitmap,  /* OUT */  
    hpss_server_attr_t    SvrAttrIn,      /* IN */  
    hpss_server_attr_t    SvrAttrOut,     /* OUT */  
    trpc_status_t        RPCError);       /* OUT */
```

Description

The **mvr_ServerSetAttrs** function is called to alter attribute values for the current state of the Mover's object. The attributes which may be set are:

- *RegisterBitmap.*
- *AdministrativeState.*

ST_SHUTDOWN

ST_HALT

ST_REINITIALIZED

ST_REPAIRED

Parameters

| | |
|------------------------|--|
| <i>Binding</i> | The TRPC binding handle. |
| <i>CNH</i> | Pointer to an HPSS connection handle. |
| <i>RequestID</i> | Request ID for this query. |
| <i>InSelectBitmap</i> | Indicates which server state attributes are to be changed. |
| <i>OutSelectBitmap</i> | Indicated which server state attributes were change as a result of this request. |
| <i>SvrAttrIn</i> | Pointer to a server_attr_t structure that contains the desired new server state attribute values. |
| <i>SvrAttrOut</i> | Pointer to a server_attr_t structure to hold the attribute values after the operation. |
| <i>RPCError</i> | Pointer to status location; will contain RPC error indication. |

Return values

Upon successful completion, a value of zero (0) is returned. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|---------------|---|
| HPSS_EAGAIN | The Mover is currently in the process of shutting down. |
| HPSS_EINVAL | Invalid attribute value. |
| HPSS_EBADCONN | Connection handle is not valid for this request. |
| HPSS_EPERM | Client not authorized for this request. |

See also

mvr_ServerGetAttrs.

Clients

Storage System Management.

Notes

The **hpss_server_attrib_t** structure is the common HPSS server object attribute structure.

5.1.14. mvr_Write

Purpose

Write data to a device or devices controlled by the Mover.

Syntax

```
#include "hpss_iod.h"

signed32
mvr_Write(
    IOD_t *IOD,      /* IN */
    IOR_t *IOR);    /* OUT */
```

Description

The **mvr_Write** function is called to receive data from another Mover (either HPSS or client Mover) and write that data to devices controlled by the Mover.

Parameters

| | |
|--------------------------------|--|
| <i>IOD->RequestID</i> | Request identifier. |
| <i>IOD->Function</i> | Set to IOD_WRITE . |
| <i>IOD->Flags</i> | Processing options (see IOD/IOR design specification for details). Valid values include REPLYWHENREADY and LAST_IN_XFER. |
| <i>IOD->SrcDescLength</i> | Number of source descriptors. |
| <i>IOD->SinkDeskLength</i> | Number of sink descriptors. |
| <i>IOD->SrcDescList</i> | List of descriptors describing data source. |
| <i>IOD->SinkDescList</i> | List of descriptors describing data sink. |
| <i>IOR->RequestID</i> | Request identifier. |
| <i>IOR->Flags</i> | IOR_COMPLETE indicates request is complete. IOR_ERROR indicates an error was encountered. |
| <i>IOR->Status</i> | Status of the request. |
| <i>IOR->SrcReplyLength</i> | Number of source reply descriptors. |
| <i>IOR->SinkReplyLength</i> | Number of sink reply descriptors. |
| <i>IOR->SrcReplyList</i> | List of descriptors describing data source status. |
| <i>IOR->SinkReplyList</i> | List of descriptors describing data sink status. |

Return values

A value of zero (0) is returned upon successful completion of the write request. If an error is detected, a value is returned that indicates the specific error.

Error conditions

| | |
|-----------------|--|
| HPSS_EABORT | Request was aborted. |
| HPSS_EBADF | Invalid device identifier. |
| HPSS_EOM | End of media reached during write. |
| HPSS_EINVAL | An offset or resulting offset would exceed the capability of the device. |
| HPSS_EIO | An I/O error occurred. |
| HPSS_EWRPROTECT | Device indicated is read-only (is this error standard enough - it is not POSIX, but is in AIX). |
| HPSS_EOPNOTSUPP | Requested operation is not supported (e.g., IPI-3 data transfer request not supported by Mover). |
| HPSS_ERANGE | A value specified in the IOD is out of range. |
| HPSS_ESYSTEM | Operating system service failed. |
| HPSS_ETRANSFER | Transfer of data from client failed. |

See also

mvr_Read.

Clients

Storage Server.

Notes

None.

5.2. Data Definitions

This section describes key internal data definitions and all externally used data definitions which are provided by this subsystem. A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

5.2.1. **Mover State Structure - mover_attr_t**

Description

The Mover State Structure describes the current state of the Mover. This structure contains all of the information that can be obtained and/or modified through the System Management interface. Fields in the structure contain the overall Mover state, and Mover statistics.

Format

The Mover State Attribute Structure has the following format:

```
typedef struct mover_attr {
    unsigned32    NumberOfRequestTasks;
    unsigned32    NumberOfActiveRequests;
    unsigned32    NumberOfRequestsProcessed;
    unsigned32    NumberOfDataTransfers;
    unsigned32    NumberOfRequestErrors;
    signed32     BufferSize;
    u_signed64    NumberOfBytesMoved;
    u_signed64    RegisterBitmap;
    timestamp_t   TotalMoveTime;
    timestamp_t   TimeOfLastGetState;
    timestamp_t   TimeOfLastStatReset;
} mover_attr_t;
```

NumberOfRequestTasks

This field contains the number of tasks that are currently processing requests for this Mover. This field can be only be queried (i.e., cannot be set).

NumberOfActiveRequests

This field contains the number of I/O requests that are currently active. This field may be used to determine whether the Mover has completely suspended or is waiting for requests to complete.

NumberOfRequestsProcessed

This field contains the number of requests that the Mover has processed since it was initialized, or the Mover statistics were reset.

NumberOfDataTransfers

This field contains the number of data transfers that the Mover has processed since it was initialized, or the Mover statistics were reset.

NumberOfRequestErrors

This field contains the number of requests that have encountered errors since the Mover was initialized or the Mover statistics were reset.

BufferSize

This field contains the size of the buffers to be used by the Mover to perform double buffering during data transfers.

NumberOfBytesMoved

This field contains the number of bytes that have been transferred by the Mover.

TotalMoveTime

This field contains the amount of elapsed time taken to transfer the *NumberOfBytesMoved* bytes that have been transferred by the Mover.

TimeOfLastGetState

This field contains the time last that a client queried the state of the Mover. This field can only be queried.

TimeOfLastStatReset

This field contains the time that the Mover statistics were last reset. This field can only be queried.

Clients

The following clients access the data definition:

Storage System Management.

5.2.2. Device Descriptor - devdesc_attr_t**Description**

The Device Descriptor contains information relating to the devices that the Mover controls.

Format

The Device Descriptor has the following format:

```
typedef struct devdesc_attr {
    device_desc_md_t      DevDescMetaData;
    device_desc_record_t  DevDescRecord;
} devdesc_attr_t;
```

DevDescMetaData

This field contains the persistent attributes for a device.

```
typedef struct device_desc_md {
    signed32      Version;
    char          DeviceName[64];
    hpss_media_t DeviceType;
    unsigned32    DeviceID;
    unsigned32    Flags;
    unsigned32    SecurityLabel;
    unsigned32    MediaBlockSize;
    signed32      OperationalState;
    signed32      UsageState;
    signed32      AdministrativeState;
    u_signed64    RegisterBitmap;
    u_signed64    NumberOfBytes;
    uuid_t        MvrID;
} device_desc_md_t;
```

Version

This field contains the version number of the device metadata structure.

DeviceName

This field contains the pathname of the device.

Device Type

This field contains the type of the device.

DeviceID

This field contains the Mover device identifier associated with this device.

Flags

This field contains flags indicating the default state and capabilities for the device. Valid values include:

MVR_DEV_READY - Device is allowed to handle requests.

MVR_DEV_READ_ALLOWED - Device allows read requests.

MVR_DEV_WRITE_ALLOWED - Device allows write requests.

MVR_DEV_WRITE_ONCE - WORM device.

MVR_DEV_MOUNTABLE - Device supports removable media.

MVR_DEV_LOCATE_SUPPORT - Device supports absolute positioning.

MVR_DEV_COMPRESS_SUPP - Device support data compression.

MVR_DEV_IPI3_SUPP - Device supports IPI-3 third party data transfers.

MVR_DEV_WTM_ZERO_TO_SYNC - Device supports a write tapemark command with zero count to synchronize device buffers to the media.

MVR_DEV_NDELAY_SUPP - Device supports opening with O_NDELAY set, followed by subsequent I/O commands.

MVR_DEV_SHARED_USE - Mover should allow multiple Mover tasks to command the device simultaneously (anticipated usage for disk devices).

SecurityLabel

This field contains the security label associated with the device.

OperationalState

This field contains the current operational state (enabled, disabled, suspect, etc.) of the device.

UsageState

This field contains the current usage state (idle, active, busy, etc.) of the device.

AdministrativeState

This field contains the current administrative state of the device.

RegisterBitmap

This field contains a bitmap that indicates which device attributes are registered for notification of value changes.

NumberOfBytes

This field contains the number of bytes that the device contains. If this value is not known (e.g., for tape devices) this value will have all bits turned on.

DevDescRecord

This field contains the volatile attributes for a device, that will be reinitialized during Mover initialization.

```
typedef struct device_desc_record {
    unsigned32      State;
    unsigned32      SetState;
    unsigned32      VolFlags;
    unsigned32      NumberOfErrors;
    unsigned32      BlockSize;
    unsigned32      AuxAddrInfo;
    signed32        DeviceFD;
    signed32        TaskID[64];
    char            VolumeID[8];
    u_signed64      BytesRead;
    u_signed64      BytesWritten;
    positiondesc_t  Position;
} device_desc_record_t;
```

State

This field contains the current state of the device. It cannot be set; state changed should be made by setting the *SetState* field. Valid values include:

MVR_DEV_MOUNTED - If this bit is set, media is currently mounted on the device.

MVR_DEV_OPEN_READ - If this bit is set, the device is opened for reading.

MVR_DEV_OPEN_WRITE - If this bit is set, the device is open for writing.

MVR_DEV_INUSE - If this bit is set, the device is currently in use by a Mover task.

MVR_DEV_ERROR - If this bit is set, the device is in an error condition and is not ready for use.

MVR_DEV_EOT - If this bit is set, the media is currently at EOT.

MVR_DEV_MIDBLOCK - If this bit is set, the Mover's position of this device is logically in the middle of a media block.

MVR_DEV_UNSYNCED - If this bit is set, there has been data written to the device that is not guaranteed to have been flushed to the media.

SetState

This field contains indication of a new state to which the Mover has been requested to change. Valid values include:

MVR_DEV_TOGGLE_READ - If this bit is set, the MVR_DEV_READALLOWED bit in the *State* field is toggled.

MVR_DEV_TOGGLE_WRITE - If this bit is set, the MVR_DEV_WRITEALLOWED bit in the *State* field is toggled.

MVR_DEV_TOGGLE_INUSE - If this bit is set, the **MVR_DEV_INUSE** bit in the *State* field is toggled.

MVR_DEV_RESET_NUMERROR - If this bit is set, the *NumberOfErrors* field will be reset to zero.

MVR_DEV_RESET_BYTESREAD - If this bit is set, the *BytesRead* field will be reset to zero.

MVR_DEV_RESET_BYTESWRITTEN - If this bit is set, the *BytesWritten* field will be reset to zero.

MVR_DEV_LOCKED - If this bit is set, the device will be locked.

MVR_DEV_UNLOCKED - If this bit is set, the device will be unlocked.

MVR_DEV_REPAIRED - If this bit is set, the device's *OperationalState* will be reset.

VolFlags

This field contains flags which describe the media format currently associated with the media loaded on the drive. Valid format types include **MVR_DEV_HPSS_VOL** (HPSS format) and **HPSS_DEV_UNITREE_VOL** (NSL UniTree format). For UniTree formats, another flag - **MVR_DEV_VOL_USE_BLK_HDRS** - is also supported (indicates whether tape contains per block headers).

NumberOfErrors

This field contains the number of error that have been encountered on the device.

BlockSize

This field contains the current size of the blocks that will be written to the device.

AuxAddrInfo

This field contains the number of blocks to be written between tapemarks on the device for tape; starting data block information for disk.

DeviceFD

This field contains the device driver associated with the open device. If the device is not currently open, this field contains a value of negative one (-1).

TaskID

This field contains an array identifying the Mover tasks that currently control the device.

VolFlags

This field contains the volume label for the media currently loaded on the device.

BytesRead

This field contains the number of bytes that have been read from the device.

BytesWritten

This field contains the number of bytes that have been written to the device.

Position

This field contains the current relative and absolute device positioning information, if available.

Setting this field causes the media to be repositioned.

```
typedef positiondesc {
    signed32          Whence;
    u_signed64       Granularity;
    relative_address_t RelativePosition;
    absolute_address_t AbsolutePosition;
} position_desc_t;
```

Whence

This field describes the origin to use in position the device. Valid values are (0) for an origin from the beginning of the media, and (1) for an origin from the current position of the media.

Granularity

This field contains the granularity at which to interpret the *Offset* field within the relative position structure.

RelativePosition

This field contains relative positioning information, including the partition, section and offset.

```
typedef struct relative_address {
    byte          Version;
    byte          Reserved;
    unsigned16    Partition;
    unsigned32    Section;
    u_signed64    Offset;
} relative_address_t;
```

Version

This field contains the version number of the relative position structure.

Reserved

This field is included for alignment purposes only.

Partition

This field contains the partition number, for devices that support multiple partitions per volume (e.g., DD2).

Section

This field contains the section number, which will indicate the number of tapemarks to be skipped.

Offset

This field contains the byte offset within the section, which will be used to determine the exact position within a section, including any blocks which may have to be skipped.

AbsolutePosition

This field contains absolute positioning information, if available.

```
typedef struct absolute_address {
    byte  AbsAddr[4];
} absolute_address_t;
```

AbsAddr

This field contains the device specific positioning information. This field can be used to hold tach counts for 3480/3490 type devices used for high speed positioning. A value of zero (0) indicates that absolute positioning information is not present.

Clients

The following clients access the data definition:

Storage Server, Storage System Management.

5.2.3. Mover Configuration Structure - mvr_config_t

Description

The Mover Configuration Structure contains the information which the Mover must be able to access to perform successful initialization. This information will be stored in a Mover specific configuration metadata file..

Format

The Mover Configuration Structure has the following format:

```
typedef mvr_config {
    uuid_t      MvrID;
    u_signed64 MoverRegisterBitmap;
    u_signed64 GenericDeviceRegisterBitmap;
    u_signed64 EncryptionKey;
    signed32    MvrBufferSize;
    idl_char    DeviceConfigFileName[HPSS_MAX_PATH_NAME];
    idl_char    MvrTcpPathName[HPSS_MAX_PATH_NAME];
    idl_char    MvrHostname[HPSS_MAX_HOSTNAME];
    unsigned16 MvrTcpPort;
    unsigned16 Reserved1;
} mvr_config_t;
```

MvrID

This field contains the ID of the server to which this entry pertains.

MoverRegisterBitmap

This field contains the initial SSM registration bitmap for the Mover managed object.

GenericDeviceRegisterBitmap

This field contains the SSM registration bitmap for the Mover generic device. This value will be ORed with each device's individual SSM registration bitmap to determine if a notification should be generated.

EncryptionKey

This field contains an encryption key used in validating connections established with the Mover.

MvrBufferSize

This field contains the size to be used for the buffers utilized by the Mover to perform double buffering.

DeviceConfigFileName

This field contains the name of the file that contains the devices defined for this Mover.

MvrTcpPathName

This field contains the pathname of the program which runs the Mover's IOD interface code.

MvrHostname

This field contains the host interface name that the Mover should use for listening for connection for its IOD interface.

MvrTcpPort

This field contains the IP port number on which the Mover is to listen for connections for its IOD interface.

Reserved1

This field is unused, and is included for padding purposes only.

Clients

The following clients access the data definition:

Mover, Storage System Management.

5.2.4. Mover Protocol Message Structures**Description**

The Mover protocol utilizes a number of structures that are passed between the active and passive sides of a data transfer. An initiator structure is used to relay transfer window and addressing information. A completion structure is used to relay status information about a piece of the transfer. Address structures are used to relay specific endpoint addressing information.

Format

The Mover protocol initiator message has the following format:

```
typedef struct initiator_msg {
    u_signed64 Delimiter;
    unsigned32 Flags;
    unsigned32 Type;
    u_signed64 Offset;
    u_signed64 Length;
    u_signed64 BlockSize;
    u_signed64 StripeWidth;
    u_signed64 Stride;
    u_signed64 TotalLength;
    char SecurityTicket[8];
    u_signed64 CheckSum;
} initiator_msg_t;
```

Delimiter

Contains a fixed value, used to mark the beginning of the message.

Flags

A bit vector containing transfer options. Valid values are:

| | |
|---------------------------------|--|
| MVRPROT_RESPONDER - | sender of message will be responder. |
| MVRPROT_ADDR_FOLLOWS - | an address message will immediately follow this message. |
| MVRPROT_COMP_REPLY - | the initiator requests a completion message. |
| MVRPROT_HOLD_RESOURCES - | keep connection open after current piece of transfer has been completed. |

Type

Contains the data transfer mechanism identifier. Valid values are:

NET_ADDRESS - transfer via TCP/IP.

IPI_ADDRESS - transfer via IPI-3 over HIPPI.

SHM_ADDRESS - transfer via a shared memory segment.

Offset

Contains the transfer offset for this piece of the data transfer.

Length

Contains the length of this piece of the data transfer.

BlockSize

Currently unused (for future exchange of striping information).

StripeWidth

Currently unused (for future exchange of striping information).

Stride

Currently unused (for future exchange of striping information).

TotalLength

Contains the total length, from the current transfer offset, for which the sender will control the transfer. This information is used to optimize passive side read operations (during internal HPSS data transfers).

SecurityTicket

Currently unused.

Checksum

A checksum calculated for this message, to verify correct message transmission.

The Mover protocol completion message structure has the following format:

```
typedef struct completion_msg {
    u_signed64 Delimiter;
    unsigned32 Flags;
    unsigned32 Status;
    u_signed64 BytesMoved;
    char      SecurityTicket[8];
    u_signed64 CheckSum;
} completion_msg_t;
```

Delimiter

Contains a fixed value, used to mark the beginning of the message.

Flags

Currently unused.

Status

Contains the completion status for the current piece of the transfer.

BytesMoved

Contains the number of bytes successfully transferred.

SecurityTicket

Currently unused.

Checksum

A checksum calculated for this message, to verify correct message transmission.

The Mover protocol network address structure has the following format:

```
typedef struct initiator_ipaddr {
    u_signed64    Delimiter;
    unsigned32    Flags;
    netaddress_t  IpAddr;
    char          SecurityTicket[8];
    u_signed64    CheckSum;
} initiator_ipaddr_t;
```

Delimiter

Contains a fixed value, used to mark the beginning of the message.

Flags

Currently unused.

IpAddr

The TCP/IP address information (see the IOD/IOR Design Specification for details).

SecurityTicket

Currently unused.

Checksum

A checksum calculated for this message, to verify correct message transmission.

The Mover protocol IPI-3 address structure has the following format:

```
typedef struct initiator_ipi3addr {
    u_signed64    Delimiter;
    unsigned32    Flags;
    ipiaddress_t  Ipi3Addr;
    char          SecurityTicket[8];
    u_signed64    CheckSum;
} initiator_ipi3addr_t;
```

Delimiter

Contains a fixed value, used to mark the beginning of the message.

Flags

Currently unused.

Ipi3Addr

The IPI-3 address information (see the IOD/IOR Design Specification for details).

SecurityTicket

Currently unused.

Checksum

A checksum calculated for this message, to verify correct message transmission.

The Mover protocol shared memory segment address structure has the following format:

```
typedef struct initiator_shmaddr {
    u_signed64    Delimiter;
    unsigned32    Flags;
    shmaddress_t  ShmAddr;
    char          SecurityTicket[8];
    u_signed64    CheckSum;
} initiator_shmaddr_t;
```

Delimiter

Contains a fixed value, used to mark the beginning of the message.

Flags

Currently unused.

ShmAddr

The shared memory segment information (see the IOD/IOR Design Specification for details).

SecurityTicket

Currently unused.

Checksum

A checksum calculated for this message, to verify correct message transmission.

Clients

The following clients access the data definition:

Mover, Client API.

6. Physical Volume Library Functions

This chapter specifies the Physical Volume Library programming interface. Specifically, the following information is provided:

Application Programming Interfaces (APIs)

Data Definitions

6.1. API Functions

This section describes all APIs which are provided for use by another HPSS subsystem or by a client external to HPSS. The API interface specification includes the following information:

Name

Syntax

Description

Parameters

Return Values

Error Conditions

Related Information

Clients

Notes

6.1.1. pvl_AllocateVol

Purpose

Allocate a volume to a particular client.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_AllocateVol(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    media_type_t      *Media,    /* IN */  
    uuid_t            *PVR,      /* IN */  
    vol_t             *Volume);  /* IN/OUT */
```

Description

If the volume is currently in the scratch state, the volume is allocated to the client calling the function.

Parameters

| | |
|---------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Media</i> | Media type of volume to allocate if the volume argument is NULL_CART. If media is NULL and no specific volume is given, an arbitrary scratch volume will be allocated to the client. |
| <i>PVR</i> | PVR from which to select a volume when a specific volume is not allocated. May be left NULL to get a cartridge from any PVR. Will be ignored when a specific volume is allocated. |
| <i>Volume</i> | Specific volume to allocate. When specified, it overrides the <i>Media</i> argument and the <i>PVR</i> argument. If volume is not specified then the <i>Media</i> and <i>PVR</i> are used to determine which volume to allocate. |

The ID of the allocated volume is returned to the client in this field regardless of the method used to select the volume.

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of volume is undefined.

Error conditions

| | |
|---------------|--|
| HPSS_ENOTYET | Server administrative set not yet Unlocked. |
| HPSS_EALREADY | Volume is already allocated to client. |
| HPSS_EINVAL | The volume is not currently in a scratch state. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOENT | No scratch volume found which matches the request. |

HPSS_EPERM

Generic volume ID invalid for this operation.

See also

pvl_DeallocateVol.

Clients

Storage Server.

Notes

Selection of the volume by media type and/or PVR will not be supported in the current HPSS because scratch pools are not supported.

6.1.2. `pvl_CancelAllJobs`

Purpose

Cancels all jobs associated with a specific connection handle.

Syntax

```
#include "pvl_interface.h"

signed32 pvl_CancelAllJobs(
    handle_t          Bh,    /* IN */
    hpss_connect_handle_t *Ch); /* IN */
```

Description

Releases all PVL resources held by the client making the call (for example, cancels pending mounts, mounts which have actually occurred, pending imports and exports, synchronous mounts, etc.).

This function will be configured as the RPC rundown routine in the PVL server initialization code.

Parameters

| | |
|-----------|----------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|-----------------------------|
| HPSS_ENOENT | No job entry was not found. |
|-------------|-----------------------------|

See also

`pvl_MountNew`, `pvl_MountCommit`, `pvl_Move`, `pvl_Import`, `pvl_Export`.

Clients

Storage Server, RPC Rundown Routine.

Notes

This function will dismount any volumes which are part of a job with which the client was involved. This includes volumes that were requested by other clients.

The job will remain in the queue while any mounted volumes are being dismounted.

Any notifications pending for the canceled job will be sent with an error code indicating that the job was canceled. This includes notifications to clients that had volumes included as part of a job that is being canceled.

6.1.3. pvl_CreateDrive

Purpose

Create new drive entry.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_CreateDrive(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    drive_data_t      *DriveData); /* IN */
```

Description

Create metadata for a new drive and add the drive to the PVL's tables.

Parameters

| | |
|------------------|----------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>DriveData</i> | New drive to create. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|--|
| HPSS_EINVAL | Fixed PVL drive table exceeded or PVR specified in the input drive information does not exist. |
| HPSS_EEXIST | Drive already exists. |
| HPSS_EMDM | Metadata manager failure. |

See also

pvl_DeleteDrive.

Clients

Storage System Manager.

Notes

A maximum of 10 new drives can be created between restarts of the PVL with this API. Drives may be created directly in metadata while the PVL is not running..

6.1.4. `pvl_DeallocateVol`

Purpose

Return a volume to the scratch pool.

Syntax

```
#include "pvl_interface.h"

signed32 pvl_DeallocateVol(
    handle_t          Bh,      /* IN */
    hpss_connect_handle_t Ch, /* IN */
    vol_t            Vol); /* IN */
```

Description

If the volume is currently allocated by the client calling `pvl_DeallocateVol`, then the volume is returned to the scratch pool.

Parameters

| | |
|------------|---------------------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Vol</i> | Volume to return to the scratch pool. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|---|
| HPSS_ENOTYET | Server administrative set not yet Unlocked. |
| HPSS_EALREADY | Volume is already deallocated(scratch). |
| HPSS_EOWNER | Volume not currently allocated to client. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_EPERM | Generic volume ID invalid for this operation. |

See also

`pvl_AllocateVol`.

Clients

Storage Server, Storage System Manager.

Notes

All volumes on a cartridge must be deallocated before a cartridge can be exported unless the client calling `pvl_Export` is the client that allocated the volumes on the cartridge.

6.1.5. pvl_DeleteDrive

Purpose

Delete drive entry.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_DeleteDrive(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    drive_t          DriveID);    /* IN */
```

Description

Delete drive from the PVL's tables.

Parameters

| | |
|----------------|----------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>DriveID</i> | Drive to delete. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------|---|
| HPSS_EBUSY | Operational state is not disabled or the drive is currently busy. |
| HPSS_EXIST | Drive does not exist. |
| HPSS_EMDM | Metadata call failure. |

See also

pvl_CreateDrive.

Clients

Storage System Manager.

Notes

None.

6.1.6. `pvl_DismountDrive`

Purpose

Forces the dismount of a specified drive.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_DismountDrive(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    drive_t          Drive);     /* IN */
```

Description

Dismount any physical volume in a drive. This function will probably be called by the SSM if the SS fails to dismount some volumes. It will only be used for error recovery.

Parameters

| | |
|--------------|----------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Drive</i> | Drive to dismount. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|--|
| HPSS_EOWNER | Client is not registered with job ID. Does not apply to SSM. |
| HPSS_EEXIST | Invalid drive ID specified. |
| HPSS_EBUSY | Drive specified is currently dismounting a cartridge. |

See also

`pvl_CancelAllJobs`, `pvl_DismountJobId`, `pvl_DismountVolume`.

Clients

Storage System Manager.

Notes

Any asynchronous notifications pending for the canceled job will be sent with an error code indicating that the job was canceled. This includes asynchronous notifications to clients that had volumes included as part of a job that is being canceled.

The job remains in the queue while any mounted volumes are being dismounted.

6.1.7. pvl_DismountJobId**Purpose**

Dismounts all volumes associated with a specific job.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_DismountJobId(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    job_id_t         JobId);     /* IN */
```

Description

Dismount all physical volumes associated with a job. The job is canceled regardless of the state it is in when `pvl_DismountJobId` is called. The job is canceled even if **pvl_MountCommit** has not been called.

This function can also be used to cancel a pending Import, Export, or Move operation.

Parameters

| | |
|--------------|----------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>JobId</i> | Job ID. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|---------------------------------------|
| HPSS_EEXIST | Job ID does not exist. |
| HPSS_EOWNER | Client is not registered with job ID. |

See also

pvl_Mount, **pvl_MountNew**, **pvl_MountCommit**, **pvl_Move**, **pvl_Import**, **pvl_Export**, **pvl_DismountVolume**, **pvl_DismountDrive**.

Clients

Storage Server, Storage System Manager.

Notes

No mount optimization is done currently. If a dismount is requested for a volume and a mount request for the volume is pending, the volume may be dismounted and remounted. This is terribly inefficient, but shouldn't matter much for the current release since we have one Storage Server that won't dismount a volume until it is done with the volume.

Any notifications pending for the canceled job will be sent with an error code indicating that the job was canceled. This includes notifications to clients that had volumes included as part of a job that is being canceled.

The job remains in the queue while any mounted volumes are being dismounted.

6.1.8. `pvl_DismountVolume`

Purpose

Dismounts a single volume.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_DismountVolume(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    job_id_t         JobId,      /* IN */  
    vol_t           *Vol);      /* IN */
```

Description

Dismount a specific physical volume. This function can be used to remove a volume from a job that has not been committed. It can also be used to dismount a single volume from a set of mounted volumes if the client is done using that particular volume.

Parameters

| | |
|--------------|----------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>JobId</i> | Job ID. |
| <i>Vol</i> | Volume to dismount. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|---------------------------------------|
| HPSS_ENOENT | Volume not part of job ID. |
| HPSS_EEXIST | Job ID does not exist. |
| HPSS_EOWNER | Client is not registered with job ID. |

See also

`pvl_Mount`, `pvl_MountNew`, `pvl_MountAdd`, `pvl_MountCommit`, `pvl_DismountJobId`, `pvl_DismountDrive`.

Clients

Storage Server.

Notes

No notifications will be sent even if some are pending for the volume.

The job remains in the queue while any mounted volumes are being dismounted.

6.1.9. pvl_DriveGetAttrs

Purpose

Get the current values of the attributes of a drive.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_DriveGetAttrs(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    drive_t          Drive,       /* IN */  
    drive_data_t     *DriveData); /* OUT */
```

Description

Returns the value of all Drive attributes for the drive specified by the *Drive* argument.

Parameters

| | |
|------------------|-------------------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Drive</i> | Drive to query. |
| <i>DriveData</i> | Attributes of Drive Managed Object. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of *DriveData* is undefined.

Error conditions

| | |
|-------------|-----------------------------|
| HPSS_EEXIST | Invalid drive ID specified. |
|-------------|-----------------------------|

See also

pvl_DriveSetAttrs.

Clients

Storage System Manager.

Notes

None.

6.1.10. pvl_DriveSetAttrs

Purpose

Set the current values of the attributes of a drive.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_DriveSetAttrs(  
    handle_t                Bh,                /* IN */  
    hpss_connect_handle_t  *Ch,                /* IN */  
    u_signed64             InSelectBitmap,     /* IN */  
    drive_data_t          *InDriveData,       /* IN */  
    u_signed64             *OutSelectBitmap,   /* OUT */  
    drive_data_t          *OutDriveData);     /* OUT */
```

Description

Sets drive attributes to the value of the corresponding field of the attributes argument. Only those attributes identified by the *InSelectBitmap* field are set.

Parameters

| | |
|------------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>InSelectBitmap</i> | Bitmap indicates which object attributes are to be set. |
| <i>InDriveData</i> | New values of attributes to be set. |
| <i>OutSelectBitmap</i> | Bitmap indicates which attributes were actually modified. |
| <i>OutDriveData</i> | Complete managed object including the updated fields. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|---|
| HPSS_EINVAL | Attempted to set a read-only attribute. |
| HPSS_EEXIST | Invalid drive ID specified. |
| HPSS_EBUSY | Unable to set attribute, drive is busy. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_EAUTH | Client is not authorized to set drive(not SSM or not the PVR in which drive resides). |

See also

pvl_DriveGetAttrs.

Clients

Storage System Manager.

Notes

If the *RegisterBitmap* field is set, it will be stored in metadata and remain set across PVL restarts.

If the *Drive* contains the ID of the generic drive object (PVL_GENERIC_DRIVE_ID) then only the *RegisterBitmap* can be set. This bitmap will be ORed with the *RegisterBitmap* attribute of each drive object to determine if a notification is to be sent to the SSM.

The following table indicates which fields can be modified through this function:

| | |
|-----|----------------------|
| NO | DriveID |
| NO | Version; |
| YES | RegisterBitmap |
| NO | PVR |
| NO | DriveAddress |
| NO | DriveType |
| NO | DriveInfo |
| YES | MaintenanceDate |
| NO | AllocatedClientID |
| NO | MountedVolume |
| YES | MountsSinceLastMaint |
| YES | OperationalState |
| NO | UsageState |
| YES | AdministrativeState |
| NO | DriveState |
| NO | MvrHost |
| NO | MvrPort |

6.1.11. `pvl_Export`

Purpose

Exports a cartridge from the HPSS system.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_Export(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    cart_t           *Cart);      /* IN */
```

Description

Remove all information about the specified cartridge from the PVL and PVR. The cartridge is physically exported from the system.

Parameters

| | |
|-------------|----------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Cart</i> | Cartridge to export. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--|
| HPSS_EINVAL | Attempted to export the generic cartridge record. |
| HPSS_EOWNER | One or more volumes on the cartridge is not allocated to client. |
| HPSS_ENOTYET | Server administrative state not yet Unlocked. |
| HPSS_EEXPORT | Export already pending for cartridge. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOENT | Cartridge not found. |
| HPSS_ENOMOUNT | Unable to eject the cartridge. |

See also

`pvl_Import`.

Clients

Storage System Manager.

Notes

A cartridge can be exported if all volumes are in the SCRATCH state or if the client requesting the export is the client that has all volumes allocated. Even the Storage System Manager will not be allowed to export a volume allocated by another client. The Storage System Manager must first scratch the volume or force the client to do so.

6.1.12. pvl_Import

Purpose

Imports a new cartridge or fixed media(disk) into an HPSS system.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_Import(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    media_type_t      *Media,       /* IN */
    cart_t            *Cart,        /* IN */
    unsigned32        ImportType,  /* IN */
    side_t            Sides,       /* IN */
    uuid_t            *Pvr,         /* IN */
    location_t        *Location,    /* IN */
    manufacturer_t    *Manu,       /* IN */
    lot_number_t      *Lot);       /* IN */
```

Description

Import a cartridge or fixed media into the HPSS system. An internal label is written on each volume of the media . The date entered into service is set to the current date and time. The external label on the cartridge is verified if possible.

Parameters

| | | | | | |
|-------------------|---|---------|---|---------|--|
| <i>Bh</i> | RPC binding handle. | | | | |
| <i>Ch</i> | HPSS connect handle. | | | | |
| <i>Media</i> | Media type of the cartridge to be imported. | | | | |
| <i>Cart</i> | External label on the cartridge. | | | | |
| <i>ImportType</i> | Type of import. One of: <table> <tr> <td>SCRATCH</td> <td>The media will be imported unless it has an ANSI or HPSS label which is different than the <i>Cart</i> specified.</td> </tr> <tr> <td>DEFAULT</td> <td>The media will be imported only if the ANSI or HPSS label is the same as that specified as input or the media has no data(two tapemarks at the start of the tape).</td> </tr> </table> | SCRATCH | The media will be imported unless it has an ANSI or HPSS label which is different than the <i>Cart</i> specified. | DEFAULT | The media will be imported only if the ANSI or HPSS label is the same as that specified as input or the media has no data(two tapemarks at the start of the tape). |
| SCRATCH | The media will be imported unless it has an ANSI or HPSS label which is different than the <i>Cart</i> specified. | | | | |
| DEFAULT | The media will be imported only if the ANSI or HPSS label is the same as that specified as input or the media has no data(two tapemarks at the start of the tape). | | | | |
| <i>Sides</i> | Number of sides (physical volumes) on the <i>Cart</i> (PVR). | | | | |
| <i>Pvr</i> | PVR that has the new <i>Cart</i> . | | | | |
| <i>Location</i> | Location of <i>Cart</i> in <i>Pvr</i> (defined by the PVR). | | | | |
| <i>Manu</i> | String identifying the manufacturer of <i>Cart</i> (client defined). | | | | |
| <i>Lot</i> | String identifying the manufacturing or purchase lot of <i>Cart</i> (client defined). | | | | |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|--|
| HPSS_EEXIST | Cartridge ID already exists in HPSS. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_EINVAL | Fixed media drive is not in the drive table or the specified PVR is not valid. |
| HPSS_ENOTYET | Server administrative set not yet Unlocked. |
| HPSS_EDISABLED | All drives of import type are disabled. |
| HPSS_ENOMOUNT | Unable to mount the cartridge to write label. |

See also

pvl_Export.

Clients

Storage System Manager.

Notes

None.

6.1.13. `pvl_Mount`**Purpose**

Synchronously mount a single volume.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_Mount(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    vol_t             *Vol,         /* IN */
    signed32          DriveOption,  /* IN */
    signed32          DriveCount,   /* IN */
    drive_t           *DriveList,  /* IN */
    job_id_t          *JobId,      /* OUT */
    drive_data_t      *DriveMounted); /* OUT */
```

Description

Mount a single volume. This function is provided as a short cut to the `pvl_MountNew`, `pvl_MountAdd`, and `pvl_MountCommit` combination of functions. `pvl_Mount` runs synchronously and may only be used when mounting a single volume, not a set of volumes.

This function might also be used by UNIX tape daemons to mount DEFAULT volumes.

Parameters

| | | | | | | | |
|---------------------|--|---------|---|-------------|--|-------------|---|
| <i>Bh</i> | RPC binding handle. | | | | | | |
| <i>Ch</i> | HPSS connect handle. | | | | | | |
| <i>Vol</i> | Volume ID to be mounted. | | | | | | |
| <i>DriveOption</i> | One of the following: <table> <tr> <td>PVL_ANY</td> <td>any drive may be used. <i>DriveList</i> is ignored.</td> </tr> <tr> <td>PVL_INCLUDE</td> <td>only drives in drive list may be used.</td> </tr> <tr> <td>PVL_EXCLUDE</td> <td>any drives <i>except</i> those in <i>DriveList</i> may be used.</td> </tr> </table> | PVL_ANY | any drive may be used. <i>DriveList</i> is ignored. | PVL_INCLUDE | only drives in drive list may be used. | PVL_EXCLUDE | any drives <i>except</i> those in <i>DriveList</i> may be used. |
| PVL_ANY | any drive may be used. <i>DriveList</i> is ignored. | | | | | | |
| PVL_INCLUDE | only drives in drive list may be used. | | | | | | |
| PVL_EXCLUDE | any drives <i>except</i> those in <i>DriveList</i> may be used. | | | | | | |
| <i>DriveCount</i> | The length of <i>DriveList</i> . | | | | | | |
| <i>DriveList</i> | The list of drives that may be (or may NOT be) used for the mount. | | | | | | |
| <i>JobId</i> | Pointer to client allocated space to return the job ID. | | | | | | |
| <i>DriveMounted</i> | Pointer to client allocated space to return the ID of the drive that was mounted. | | | | | | |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. On error, *jobId* and *driveMounted* are undefined.

Error conditions

| | |
|----------------|---|
| HPSS_EACCES | Volume not allocated to client. |
| HPSS_EEXPORT | Volume is scheduled to be exported. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOENT | Invalid volume ID specified. |
| HPSS_ENOMEM | Not enough memory. |
| HPSS_ENOMOUNT | Mount failed. |
| HPSS_ENOTYET | Server administrative set not yet Unlocked. |
| HPSS_EINVAL | Volume/Drive type mismatch. |
| HPSS_EEXIST | PVR for Volume doesn't exist. |
| HPSS_EDISABLED | All required drives are disabled. |

See also

pvl_DismountJobId, pvl_DismountVolume, pvl_DismountDrive.

Clients

UNIX Application.

Notes

None.

6.1.14. `pvl_MountAdd`**Purpose**

Add a volume to a set of volumes to be mounted for the specified job.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_MountAdd(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    job_id_t         JobId,      /* IN */
    vol_t           *Vol,        /* IN */
    signed32        DriveOption, /* IN */
    signed32        DriveCount,  /* IN */
    drive_t         *DriveList); /* IN */
```

Description

Add another volume to the list for a specific mount request.

Parameters

| | |
|--------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>JobId</i> | Job ID. |
| <i>Vol</i> | Volume ID to be mounted. |
| <i>DriveOption</i> | One of the following: |
| | PVL_ANY any drive may be used. <i>DriveList</i> is ignored. |
| | PVL_INCLUDE only drives in drive list may be used. |
| | PVL_EXCLUDE any drives <i>except</i> those in <i>DriveList</i> may be used. |
| <i>DriveCount</i> | The length of <i>DriveList</i> . |
| <i>DriveList</i> | The list of drives that may be (or may NOT be) used for the mount. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. An error only means that the volume was not added to the set. The job ID and any volumes already in the set are not affected.

Error conditions

| | |
|--------------|-------------------------------------|
| HPSS_EACCES | Volume not allocated to client. |
| HPSS_EBUSY | Cartridge already part of mount. |
| HPSS_EEXPORT | Volume is scheduled to be exported. |

| | |
|---------------|--|
| HPSS_EINVAL | Job ID already committed or job is not a mount (it is an import, export, or move). |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOENT | Invalid job ID or volume ID specified. |
| HPSS_ENOMEM | Not enough memory. |
| HPSS_ESRCH | Client is not registered with job ID. |
| HPSS_ENOTYET | Server administrative set not yet Unlocked. |
| HPSS_ETOOMANY | Not enough drives in PVR to support requested volumes. |

See also

pvl_DismountJobId, pvl_MountNew, pvl_MountCommit.

Clients

Storage Server.

Notes

None.

6.1.15. pvl_MountCommit**Purpose**

Mount a set of volumes.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_MountCommit(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    job_id_t         JobId,       /* IN */
    timestamp_t      *CommitTime); /* OUT */
```

Description

Commits the mount request associated with the job ID. All volumes should have already been added to the mount list. The time the job was committed in the PVL is returned.

Parameters

| | |
|-------------------|------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>JobId</i> | Job ID. |
| <i>CommitTime</i> | Time committed in PVL. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

A return value of 0 means that the mount request has been queued, the volumes are not actually available. As each volume mount is completed, the notify function (**ss_MountCallback**) will be called to inform the client. The *CommitTime* is used for job recovery by libpvl.

Error conditions

| | |
|--------------|--|
| HPSS_EEMPTY | No volumes have been added to the mount. |
| HPSS_EINVAL | Job ID already committed or job is not a mount (it is an import, export, or move). |
| HPSS_ENOENT | Invalid job ID specified. |
| HPSS_EOWNER | Client is not registered with job ID. |
| HPSS_ENOTYET | Server administrative set not yet Unlocked. |

See also

pvl_DismountJobId, pvl_MountNew, pvl_MountAdd.

Clients

Storage Server.

Notes

The client should be aware that the notify function (**ss_MountCallback**) can be called as soon as **pvl_MountCommit** is called, before the **pvl_MountCommit** function returns (i.e. a race condition exists).

If the client's notify function requires any set up before it can be called, the client should perform the set up when it receives the job ID from the **pvl_MountNew** function.

6.1.16. pvl_MountCompleted**Purpose**

Notify the PVL that a pending mount has completed.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_MountCompleted(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    cart_t           *Cart,       /* IN */
    side_t           Side,        /* IN */
    drive_t          Drive,       /* IN */
    job_id_t         JobId,       /* IN */
    pvl_status_t     TheStatus); /* IN */
```

Description

The PVL will verify the job, check the label on the drive, and if all is in order, complete its processing of the mount. This is intended to be used by PVRs which can report the drive a volume has been mounted on to speed up volume recognition. It is also used by the PVR to report errors in attempting to mount a cartridge.

Parameters

| | |
|------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Cart</i> | Cartridge ID. |
| <i>Side</i> | Side number (normally zero). |
| <i>Drive</i> | The ID of the drive. |
| <i>JobId</i> | Job ID of the mount which completed. |
| <i>TheStatus</i> | 0 for successful completion, non-zero to report errors. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------|---|
| HPSS_EINVAL | <i>TheStatus</i> is zero, and <i>DriveAddr</i> is not valid, OR the drive is not controlled by the PVR, OR the volume label does not match the request. |
| HPSS_EBUSY | The drive is offline, or in use by some other job. |
| HPSS_ESRCH | Cartridge/Activity/Job was not found. |
| HPSS_ESOCKET | Error occurred opening socket to Mover while attempting to read label. |
| HPSS_ESCOMM | Communication error occurred while attempting to read label. |

HPSS_ETIMEDOUT Timeout error occurred while attempting to read label.

See also

pvl_MountNew, pvl_MountCommit, pvl_MountAdd, pvl_Mount.

Clients

Physical Volume Repository.

Notes

DriveAddr and *JobId* are required if *TheStatus* is zero, indicating a successful mount. If *TheStatus* is non-zero, *DriveAddress* is ignored. If an error other than HPSS_ESRCH is returned to the PVR, the PVR will attempt to mount the cartridge in another drive in the jobs drive list.

6.1.17. pvl_MountNew**Purpose**

Begin creating a set of volumes to mount, or allow a client to add volumes to an existing set.

Syntax

```
#include "pvl_interface.h"

signed32 pvl_MountNew(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    signed32         ControllerWait, /* IN */
    job_id_t         *JobId);     /* IN/OUT */
```

Description

Returns a unique job ID that can be used to identify a set of physical volumes to be mounted in one atomic operation. Or adds a new client to the notify list for an existing job ID that has not yet been committed.

Parameters

| | |
|-----------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>ControllerWait</i> | If TRUE the PVL will wait until all the volumes can be mounted on non-conflicting controller drives. If FALSE the PVL will make an attempt to use non-conflicting controller drives, but will use any available drives. |
| <i>JobId</i> | Job ID. If NULL, this is a new mount request and a job ID will be returned. If not NULL, the function is being called by a client that is joining an existing job ID. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. On error, *jobId* is undefined.

Error conditions

| | |
|----------------|--|
| HPSS_EINVAL | Job ID already committed or job is not a mount (it is an import, export, or move). |
| HPSS_ENOENT | Nonexistent job ID specified. |
| HPSS_ENOMEM | Not enough memory. |
| HPSS_ESRCH | Client is already registered with <i>JobId</i> . |
| HPSS_ENOTYET | Server administrative state not yet Unlocked. |
| HPSS_ECONFLICT | Client is already a member of job. |

See also

pvl_DismountJobId, pvl_MountAdd, pvl_MountCommit, ss_MountCallback.

Clients

Storage Server.

Notes

In future releases, if the mount is not committed in `MOUNT_COMMIT_TIMEOUT` seconds, then the mount will automatically be aborted. This feature is not supported in HPSS the current release.

When multiple clients are involved in an atomic mount, the *ControllerWait* values are Ored together. Thus if any client wants to ensure that the job is spread across non-conflicting controllers, all volumes will be mounted in that manner.

6.1.18. pvl_Move**Purpose**

Move a cartridge from one PVR to another.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_Move(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    uuid_t           *Destination, /* IN */
    cart_t           *Cart);     /* IN */
```

Description

Moves a cartridge from one PVR to another. The entire move will be automated by the PVRs if possible. Otherwise the source PVR will use its standard eject mechanism and the destination PVR will use its standard inject mechanism.

All metadata associated with the cartridge that is under PVR control will be transferred to the new PVR.

Parameters

| | |
|--------------------|-------------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Destination</i> | PVR to receive the cartridge. |
| <i>Cart</i> | Cartridge to move. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|--|
| HPSS_ENOTYET | Server administrative set not yet Unlocked. |
| HPSS_EEXIST | Destination PVR does not exist. |
| HPSS_EEXPORT | Volume is scheduled to be exported. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENODEV | Media type not supported by destination PVR. |
| HPSS_EINVAL | Fixed media don't reside in a PVR. |
| HPSS_EDISABLED | All drives of required type in PVR are disabled. |
| HPSS_ENOENT | Cartridge not found. |
| HPSS_EPERM | Generic volume ID invalid for this operation. |
| HPSS_EALREADY | Destination and current PVR are the same. |

See also

pvl_Import, pvl_Export.

Clients

Storage System Manager.

Notes

The move will be done in the order of an inject to destination PVR, then eject from the current PVR. This order is used to prevent loss of metadata.

6.1.19. pvl_NotifyCartridge**Purpose**

Notify the PVL that a cartridge has been checked in to our out of a PVR.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_NotifyCartridge(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t Ch,      /* IN */
    signed32          Notification, /* IN */
    cart_t            Cart,        /* IN */
    uuid_t            PvrID);     /* IN */
```

Description

If the cartridge is being checked out of a PVR, and there is an outstanding mount request for any volume on that cartridge, the PVL will call the **pvr_LocateCartridge** function in each PVR that might contain the cartridge, starting with the PVR identified in *PvrID* (if any), to cause each such PVR to search for the cartridge. If another PVR reports having the cartridge, the PVL will update the location metadata to reflect the new location, cancel any outstanding mount to the previous PVR, and issue a mount to the new PVR. If the cartridge is not located by this method, the mount will remain pending in the original PVR so that the operator will be notified of the request for the tape.

If the cartridge is being checked in to a PVR, the PVL will update the metadata associated with all volumes on that cartridge to reflect the possibly new PVR containing the cartridge. If there is a pending mount request for any volume on the cartridge, the PVL will cancel the mount request to any other PVR, and issue the mount request to the new PVR.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Notification</i> | One of the following: PVR_CHECKED_IN. PVR_CHECKED_OUT. |
| <i>Cart</i> | Specific cartridge ID. |
| <i>PvrID</i> | If the PVR can determine that the cartridge has been moved to another PVR, this is the ID of the PVR to which it has moved. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of volume is undefined.

Error conditions

| | |
|--------------------|--|
| HPSS_ENOTSUPPORTED | This API is not currently implemented. |
|--------------------|--|

See also

pvr_LocateCartridge.

Clients

Physical Volume Repository.

Notes

This API is not currently called by current release PVRs nor is it implemented in the current PVL. It may be included in latter releases in conjunction with an audit capability.

6.1.20. pvl_PVLGetAttrs**Purpose**

Get the current values of the attributes of the PVL.

Syntax

```
#include "pvl_interface.h"

signed32 pvl_PVLGetAttrs(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    pvl_data_t       *PLVData);  /* OUT */
```

Description

Returns the value of all PVL attributes. All attributes have meaningful values except *ScratchVolumes* which is always set to 0.

Parameters

| | |
|----------------|--|
| <i>Bh</i> | RPC binding handle |
| <i>Ch</i> | HPSS connect handle |
| <i>PLVData</i> | Structure containing all attributes of the PVL Managed Object. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of Attributes is undefined.

Error conditions

None.

See also

pvl_PVLSetAttrs.

Clients

Storage System Manager.

Notes

None.

6.1.21. pvl_PVLSetAttrs

Purpose

Set the current values of the attributes of the PVL.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_PVLSetAttrs(  
    handle_t                Bh,                /* IN */  
    hpss_connect_handle_t  *Ch,                /* IN */  
    u_signed64              InSelectBitmap,      /* IN */  
    pvl_data_t              *InPVLData,        /* IN */  
    u_signed64              *OutSelectBitmap,    /* OUT */  
    pvl_data_t              *OutPVLData);      /* OUT */
```

Description

Sets PVL attributes to the value of the corresponding field of the attributes argument. Only those attributes identified by the *InSelectBitmap* field are set.

Parameters

| | |
|------------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>InSelectBitmap</i> | Bitmap indicates which object attributes are to be set. |
| <i>InPVLData</i> | New values of attributes to be set. |
| <i>OutSelectBitmap</i> | Bitmap indicates which attributes were actually modified. |
| <i>OutPVLData</i> | Complete managed object including the updated fields. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------------|---|
| HPSS_ENOTSUPPORTED | Attempted to set scratch volumes. Not implemented in the current release. |
| HPSS_EINVAL | Attempted to set a read-only attribute. |

See also

pvl_PVLGetAttrs.

Clients

Storage System Manager.

Notes

The *RegisterBitmap* field will not be persistent across PVL restarts.

The following table indicates which fields can be modified through this function:

| | |
|----|--------------|
| NO | <i>PVLId</i> |
|----|--------------|

NO *Version*
YES *RegisterBitmap*
YES *TotalVolumes*
YES *TotalRepositories*
NO *ScratchVolumes*
YES *TotalDrives*
YES *VolFileName*
YES *JobFileName*
YES *ActFileName*
YES *DriveFileName*
NO *PVRIDs*

6.1.22. pvl_QueueGetAttrs

Purpose

Get the current values of the attributes of the PVL request queue.

Syntax

```
#include "pvl_interface.h"

signed32 pvl_QueueGetAttrs(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t Ch,     /* IN */
    api_queue_data_t **QueueData); /* OUT */
```

Description

Returns the value of all PVL request queue attributes. The request queue contains information on all active mount, import, export, and move requests.

Parameters

| | |
|------------------|-----------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>QueueData</i> | Pointer to a list of jobs.. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of *QueueData* is undefined.

Error conditions

None.

See also

pvl_QueueSetAttrs.

Clients

Storage System Manager.

Notes

None.

6.1.23. `pvl_QueueSetAttrs`**Purpose**

Set the current values of the attributes of the PVL request queue.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_QueueSetAttrs(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    u_signed64         InSelectBitmap, /* IN */
    api_queue_data_t  *InQueueData,   /* IN */
    u_signed64         *OutSelectBitmap, /* OUT */
    api_queue_data_t  **OutQueueData); /* OUT */
```

Description

Returns the value of all PVL request queue attributes. Sets queue attributes to the value of the corresponding field of the `InQueueData` argument. Only those attributes identified by the `InSelectBitmap` field are set.

Parameters

| | |
|------------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>InSelectBitmap</i> | Bitmap indicates which object attributes are to be set. |
| <i>InQueueData</i> | New values of attributes to be set. |
| <i>OutSelectBitmap</i> | Bitmap indicates which attributes were actually modified. |
| <i>OutQueueData</i> | Complete managed object including the updated fields. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|---|
| HPSS_EINVAL | Attempted to set a read-only attribute. |
|-------------|---|

See also

`pvl_QueueGetAttrs`.

Clients

Storage System Manager.

Notes

The `RegisterBitmap` field will not be persistent across PVL restarts.

It is not possible to change the list of jobs with this function. Jobs are added via `pvl_MountAdd`, `pvl_Mount`, `pvl_Import`, etc. Jobs are removed after they are completed.

The following table indicates which fields can be modified through this function:

NO *QueueID*
NO *Version*
YES *RegisterBitmap*
NO *TotalRequests*
NO *Jobs*
NO *JobID*
NO *Next*

6.1.24. pvl_RequestGetAttrs**Purpose**

Get the current values of the attributes of a specified entry in the PVL request queue.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_RequestGetAttrs  
    handle_t                Bh,           /* IN */  
    hpss_connect_handle_t  *Ch,           /* IN */  
    job_id_t               RequestID,    /* IN */  
    request_data_t         *RequestData); /* OUT */
```

Description

Returns the value of a PVL request queue entry's attributes. An entry in the request queue contains information about a single active mount, import, export, or move request. The request queue entry to be queried is determined by the *RequestID* argument.

Parameters

| | |
|--------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>RequestID</i> | ID of specific request to query. |
| <i>RequestData</i> | Structure containing all attributes of the Request Managed Object. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of Attributes is undefined.

Error conditions

| | |
|------------|-------------------------------|
| HPSS_EXIST | Invalid request ID specified. |
|------------|-------------------------------|

See also

pvl_RequestSetAttrs.

Clients

Storage System Manager.

Notes

None.

6.1.25. pvl_RequestSetAttrs

Purpose

Set the current values of the attributes of an entry on the PVL request queue.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_RequestSetAttrs(  
    handle_t                Bh,                /* IN */  
    hpss_connect_handle_t  *Ch,                /* IN */  
    u_signed64             InSelectBitmap,      /* IN */  
    request_data_t         *InRequestData,      /* IN */  
    u_signed64             *OutSelectBitmap,     /* OUT */  
    request_data_t         *OutRequestData);    /* OUT */
```

Description

Sets request attributes to the value of the corresponding field of the *InRequestData* argument. Only those attributes identified by the *InSelectBitmap* field are set.

Parameters

| | |
|------------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>InSelectBitmap</i> | Bit map that indicates which object attributes are to be set. |
| <i>InRequestData</i> | New values of attributes to be set. |
| <i>OutSelectBitmap</i> | Bitmap indicates which attributes were actually modified. |
| <i>OutDriveData</i> | Complete managed object including the updated fields. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|---|
| HPSS_EINVAL | Attempted to set a read-only attribute. |
| HPSS_EXIST | Invalid request ID specified. |

See also

pvl_RequestGetAttrs.

Clients

Storage System Manager.

Notes

The *RegisterBitmap* field will not be persistent across PVL restarts.

The following table indicates which fields can be modified through this function:

| | |
|----|--------------|
| NO | <i>JobID</i> |
|----|--------------|

NO *Version*
YES *RegisterBitmap*
NO *RequestType*
NO *RequestTimestamp*
NO *MountedVolumes*
NO *RequestStatus*
NO *Vols*
NO *Vol*
NO *DriveRequested*
NO *Drive*
NO *State*
NO *Next*

6.1.26. pvl_ServerGetAttrs

Purpose

Get the current values of the attributes of the PVL server.

Syntax

```
#include "pvl_interface.h"

signed32 pvl_ServerGetAttrs(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    hpss_server_attr_t *ServerData); /* OUT */
```

Description

Returns the value of all PVL server attributes.

Parameters

| | |
|-------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>ServerData</i> | Structure containing all attributes of the HPSS Server Managed Object. Note that the definition of this structure is maintained in the SSM chapter. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of Attributes is undefined.

Error conditions

None.

See also

[pvl_ServerSetAttrs](#).

Clients

Storage System Manager.

Notes

None.

6.1.27. pvl_ServerSetAttrs**Purpose**

Set the current values of the attributes of the PVL server.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_ServerSetAttrs(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    u_signed64        InSelectBitmap, /* IN */
    hpss_server_attr_t *InServerData, /* IN */
    u_signed64        OutSelectBitmap, /* OUT */
    hpss_server_attr_t *OutServerData); /* OUT */
```

Description

Sets PVL server attributes to the value of the corresponding field of the *InServerData* argument. Only those attributes identified by the *InSelectBitmap* field are set.

Parameters

| | |
|------------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>InSelectBitmap</i> | Bitmap indicates which object attributes are to be set. |
| <i>InServerData</i> | New values of attributes to be set. |
| <i>OutSelectBitmap</i> | Bit map that indicates which attributes were actually modified. |
| <i>OutServerData</i> | Complete managed object including the updated fields. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------------|---|
| HPSS_ENOTSUPPORTED | Attempted to set the Administrative state to reinitialize. Not implemented in the current release. |
| HPSS_EINVAL | Attempted to set one or more read-only attributes. The <i>OutSelectBitmap</i> indicates which attributes were successfully set. |

See also

pvl_ServerGetAttrs.

Clients

Storage System Manager.

Notes

The *RegisterBitmap* field will not be persistent across PVL restarts.

The following table indicates which fields can be changed through this function:

| | |
|-----|---------------------------------|
| NO | <i>Version</i> |
| NO | <i>ServerID</i> |
| NO | <i>DescName</i> |
| NO | <i>ServerName</i> |
| NO | <i>OperationalState</i> |
| NO | <i>UsageState</i> |
| YES | <i>AdministrativeState</i> |
| NO | <i>ExecutionState</i> |
| NO | <i>ServiceStatus</i> |
| NO | <i>SecurityStatus</i> |
| NO | <i>SoftwareStatus</i> |
| NO | <i>HardwareStatus</i> |
| NO | <i>CommunicationStatus</i> |
| NO | <i>ThreadsAlarmThreshold</i> |
| NO | <i>ConnectionAlarmThreshold</i> |
| YES | <i>RegisterBitmap</i> |

6.1.28. pvl_VolumeGetAttrs**Purpose**

Get the current values of the attributes of a volume.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_VolumeGetAttrs(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t Ch,      /* IN */  
    vol_t            Vol,          /* IN */  
    vol_data_t       VolData);    /* OUT */
```

Description

Returns the value of all Volume attributes for the volume specified by the *Vol* argument.

Parameters

| | |
|----------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Vol</i> | Volume to query. |
| <i>VolData</i> | Structure containing all attributes of the Volume Managed Object. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of *VolData* is undefined.

Error conditions

| | |
|-------------|------------------------------|
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOENT | Invalid volume ID specified. |

See also

pvl_VolumeSetAttrs.

Clients

Storage System Manager.

Notes

None.

6.1.29. `pvl_VolumeSetAttrs`

Purpose

Set the current values of the attributes of a volume.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_VolumeSetAttrs(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    u_signed64        InSelectBitmap, /* IN */  
    vol_data_t        *InVolData,  /* IN */  
    u_signed64        *OutSelectBitmap, /* OUT */  
    vol_data_t        *OutVolData); /* OUT */
```

Description

Sets drive attributes to the value of the corresponding field of the *InVolData* argument. Only those attributes identified by the *InSelectBitmap* field are set.

Parameters

| | |
|------------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>InSelectBitmap</i> | Bitmap indicates which object attributes are to be set. |
| <i>InVolData</i> | New values of attributes to be set. |
| <i>OutSelectBitmap</i> | Bit map that indicates which attributes were actually modified. |
| <i>OutVolData</i> | Complete managed object including the updated fields. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|---|
| HPSS_EINVAL | Attempted to set a read-only attribute. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_EOWNER | Volume not currently allocated to client. |
| HPSS_EEXIST | Volume ID specified does not exist. |

See also

`pvl_VolumeGetAttrs`.

Clients

Storage System Manager.

Notes

If the *RegisterBitmap* field is set, it will be stored in metadata and remain set across PVL restarts.

If the *PhysicalVolumeID* field of the *InVolData* structure contains the ID of the generic volume object (*PVL_GENERIC_VOLUME_ID*) then only the *RegisterBitmap* can be set. This bitmap will be ORed with the *RegisterBitmap* attribute of each volume object to determine if a notification is to be sent to the SSM.

The following table indicates which fields can be modified through this function:

| | |
|-----|----------------------------|
| NO | <i>PhysicalVolumeID</i> |
| NO | <i>Version</i> |
| YES | <i>RegisterBitmap</i> |
| NO | <i>PVR</i> |
| NO | <i>AllocatedClientID</i> |
| NO | <i>AllocationStatus</i> |
| YES | <i>PhysicalVolumeType</i> |
| NO | <i>PhysicalVolumeLabel</i> |
| NO | <i>VolumeLabelFormat</i> |
| NO | <i>OperationalState</i> |
| NO | <i>UsageState</i> |
| NO | <i>AdministrativeState</i> |
| YES | <i>PhysicalVolumeState</i> |
| NO | <i>CartridgeID</i> |

6.1.30. `pvl_WriteVolumeLabel`

Purpose

Rewrite the internal label on a specified volume.

Syntax

```
#include "pvl_interface.h"
```

```
signed32 pvl_WriteVolumeLabel (  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    vol_t             *Vol);      /* IN */
```

Description

Rewrites the internal label on a volume. With most tape technologies this will logically erase all information on the volume. It will not physically erase any data on the volume.

Parameters

| | |
|------------|--------------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Vol</i> | Specific volume rewrite label. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of volume is undefined.

Error conditions

| | |
|----------------|--|
| HPSS_ENOTYET | Server administrative set not yet Unlocked. |
| HPSS_EINVAL | Attempted to write label on an allocated volume. |
| HPSS_EDISABLED | All drives of required type in PVR are disabled. |
| HPSS_ENOMOUNT | Unable to mount volume. |
| HPSS_ENOENT | Volume not found. |

See also

`pvl_Import`.

Clients

None.

Notes

Not currently called by a HPSS server.

6.2. Data Definitions

This section describes key internal data definitions and all externally used data definitions which are provided by this subsystem. A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

Many of the PVL data structures are linked together in a fairly complicated fashion. To make the relationships more clear, a high level view of some of the data structures is presented below.

Job List An unordered list of jobs that the PVL is currently processing. Jobs include mount, import, export, and move. A mount request will be kept on the queue from the time the **pvl_MountNew** API is issued until the time the PVR has completed dismounting all of the volumes associated with the mount request (or equivalent functions for import, export, and move commands).

Every job is given a job ID.

Next Drive List A subset of the Job List is linked in a queue. This is a queue of jobs that are waiting for drives to be assigned. The job at the head of this queue will have first shot at the next drive when it becomes available. New jobs are added to the end of this queue when they have assigned all of their cartridges. Jobs are removed from this queue after all mounts for the job are satisfied.

Note that this queue implies that a job with an earlier commit time could receive drives and be mounted before a job with an older commit time if the earlier job is assigned all of its cartridges first.

Active Cart List A list of cartridges that are currently part of one or more jobs. This list is used to quickly identify jobs that require the same cartridge. When the cartridge becomes available, all jobs waiting for the cartridge are checked and the job with the oldest commit time is assigned the cartridge.

Client Info Information about the client(s) associated with a job queue entry. The information includes the client ID and the asynchronous callback provided by the client (for operations which receive asynchronous notification).

Activity Structure Contains the status of a specific physical volume in a specific job.

The Job List, Active Cart List, and Activity Structure form a sort of 2 dimensional array.

6.2.1. PVL Data Structure - pvl_data_t

Description

Managed Object which contains PVL specific parameters. This data is maintained by the metadata manager.

Format

The PVL Data has the following format:

```
typedef struct {
    uuid_t      PVLId;
    signed32    Version;
    u_signed64  RegisterBitmap;
    signed32    TotalVolumes;
    signed32    TotalRepositories;
```

```
signed32    ScratchVolumes;  
signed32    TotalDrives;  
idl_char    VolFileName[HPSS_MAX_DCE_NAME];  
idl_char    JobFileName[HPSS_MAX_DCE_NAME];  
idl_char    ActFileName[HPSS_MAX_DCE_NAME];  
idl_char    DriveFileName[HPSS_MAX_DCE_NAME];  
uuid_t      PVRIDs[MAX_PVRS];  
} pvl_data_t;
```

PVLId

ID of the specific PVL. It is intended that there be only one PVL per HPSS installation.

Version

The number identifying the version of the PVL which understands this record.

RegisterBitmap

Bitmap that shows which job attributes the SSM has registered for change notification.

TotalVolumes

The total number of volumes currently managed by the PVL.

TotalRepositories

The number of PVRs currently controlled by the PVL.

ScratchVolumes

The number of scratch volumes currently managed by the PVL(Unused in the current release).

TotalDrives

The number of drives currently managed by the PVL.

VolFileName

The name of the SFS file containing the volume metadata.

JobFileName

The name of the SFS file containing the Job metadata.

ActFileName

The name of the SFS file containing the Activity metadata.

DriveFileName

The name of the SFS file containing the Drive metadata.

PVRIDs

The IDs of the PVRs controlled by the PVL.

Clients

The following clients access the data definition:

Physical Volume Repository, Storage System Manager, NSL UniTree Migration.

6.2.2. Queue Data Structure - api_queue_data_t

Description

The Queue Data is an external representation of the PVL's internal job list structure. The RegisterBitmap in the queue is checked each time a job is added to or removed from the queue, so that SSM can be notified of changes in the queue.

Format

The Queue Data has the following format:

```
typedef struct {
    signed32      QueueID;
    signed32      Version;
    u_signed64    RegisterBitmap;
    signed32      TotalRequests;
    job_id_list_t JobList[1];
} api_queue_data_t;
```

QueueID

There is currently only one Queue. This field is for future use.

Version

The number identifying the version of the PVL which understands this record.

RegisterBitmap

Bitmap that shows which job attributes the SSM has registered for change notification.

TotalRequests

The number of requests in the PVL's queue.

Jobs

Pointer to the first JobID structure in the queue or NULL if the queue is empty.

```
typedef struct job_id_list {
    signed32      JobID;
    signed32      RequestType;
    signed32      RequestStatus;
    signed32      MountedVolumes;
    timestamp_t   CommitTime;
    timestamp_t   RequestTimeStamp;
    struct job_id_list *Next;
} job_id_list_t;
```

JobID

The ID of a job in the queue.

RequestType

The type of job request (mount, import, etc.). Valid values are: PVL_ASYNC_MOUNT, PVL_IMPORT_DEFAULT, PVL_IMPORT_SCRATCH, PVL_DEFER_DISMOUNTS, PVL_EXPORT, PVL_MOVE, PVL_RELABEL, and PVL_SYNC_MOUNT.

RequestStatus

The status from the job request (cart wait, drive wait, etc.). Valid values are: PVL_JOB_NULL, PVL_JOB_UNCOMMITTED, PVL_JOB_CART_WAIT, PVL_JOB_DRIVE_WAIT, PVL_MOUNT_WAIT, PVL_JOB_MOUNTED, PVL_JOB_DISMOUNT_PENDING, PVL_JOB_ABORTING, PVL_JOB_INJECT, PVL_JOB_DEFER_DISMOUNTS, PVL_JOB_EJECT, PVL_JOB_IN_USE, and PVL_JOB_COMPLETED.

MountedVolumes

The number of volumes currently mounted.

CommitTime

The time the job was committed.

RequestTimeStamp

The time the job was created.

Next

This field should be null in all entries when passed through any API function calls. The field is used internally by the PVL, but will cause recursive unmarshalling if passed through an API. This may cause a stack overflow in the client.

Clients

The following clients access the data definition:

Storage System Manager.

6.2.3. PVL Job Queue Entry - request_data_t

Description

The request structure is an external representation of the internal structure of a job in the PVL's queue. Setting the RegisterBitmap in a request sets the RegisterBitmap in the corresponding job.

Format

The Request Data has the following format:

```
typedef struct {
    job_id_t      JobID;
    signed32     Version;
    u_signed64    RegisterBitmap;
    signed32     RequestType;
    timestamp_t  RequestTimestamp;
    signed32     MountedVolumes;
    signed32     RequestStatus;
    volume_list_t *Vols;
} request_data_t;
```

JobID

The unique JobID assigned by the PVL when the job was created.

Version

The number identifying the version of the PVL which understands this record.

RegisterBitmap

Bitmap that shows which job attributes the SSM has registered for change notification.

RequestType

One of the following:

PVL_MOUNT

PVL_IMPORT_SCRATCH

PVL_IMPORT_DEFAULT

PVL_EXPORT

PVL_MOVE

PVL_RELABEL

PVL_ASYNC_MOUNT

PVL_SYNC_MOUNT

PVL_DEFER_DISMOUNTS

RequestTimestamp

The time at which the job was created.

MountedVolumes

The number of volumes in the request.

RequestStatus

The state of the job, one of the following:

PVL_JOB_UNCOMMITTED

PVL_JOB_CART_WAIT

PVL_JOB_DRIVE_WAIT

PVL_JOB_MOUNT_WAIT

PVL_JOB_MOUNTED

PVL_JOB_DISMOUNT_PENDING

PVL_JOB_ABORTING

PVL_JOB_INJECT

PVL_JOB_EJECT

PVL_JOB_IN_USE

PVL_JOB_COMPLETED

PVL_JOB_DEFER_DISMOUNTS

```
typedef struct volume_list {
    vol_t          Vol;
    drive_t        Drive;
    unsigned32     State;
    signed32       DriveOption;
    signed32       DriveCount;
    drive_t        DriveList[PVL_MAX_DRIVES_PER_PVR];
    drive_type_t   DriveType;
    struct volume_list *Next;
}
```

```
} volume_list_t;
```

Vol

Volume ID of a volume which is part of the request.

Drive

Drive ID of the drive actually assigned, Zero if no drive assigned.

State

The state of this volume, one of the following:

PVL_ACT_UNCOMMITTED
PVL_ACT_CART_WAIT
PVL_ACT_CART_ASSIGNED
PVL_ACT_READING_LABEL
PVL_ACT_DRIVE_WAIT
PVL_ACT_UNLOAD_PENDING
PVL_ACT_MOUNT_PENDING
PVL_ACT_MOUNT_FAILED
PVL_ACT_MOUNTED
PVL_ACT_DISMOUNT_PENDING
PVL_ACT_INJECT
PVL_ACT_EJECT
PVL_ACT_DISMOUNTED

DriveOption

The selection option for the DriveList, one of the following:

PVL_ANY
PVL_INCLUDE
PVL_EXCLUDE

DriveCount

Count of entries in Drives. Zero to indicate any available drive.

DriveList

List of drive IDs that might be used for this request..

DriveType

Defined media types, one of the following:

MT_TAPE_DEFAULT
MT_TAPE_3480
MT_TAPE_3490
MT_TAPE_3490E
MT_TAPE_3590
MT_TAPE_DD2
MT_TAPE_8MM
MT_TAPE_4MM
MT_TAPE_VHS
MT_TAPE_4420
MT_DISK_DEFAULT
MT_DISK_9570
MT_DISK_MAXSTDKA
MT_RWOPTICAL_DEFAULT
MT_WORMOPTICAL_DEFAULT
MT_MEMORY_DEFAULT
MT_MEMORY_ZITEL
MT_TAPE_REDW
MT_TAPE_TMBRLN

Next

Pointer to the next volume in the request, NULL if this is the last volume.

Clients

The following clients access the data definition:

Storage System Manager.

6.2.4. Cartridge ID Structure - cart_t

Description

Unique identifier for a cartridge in an HPSS system. Currently, the cartridge ID is identical to the external label on the cartridge.

Format

The Cartridge ID has the following format:

```
typedef struct {
    char    Cart[CART_LABEL_LENGTH];
} cart_t;
```

Cart

ID of the cartridge. Should be identical to the external label on the cartridge. CART_LABEL_SIZE is defined as 6.

CART_LABEL_LENGTH is defined as 8, which is CART_LABEL_SIZE + 1 (for a trailing NULL) + 1 (to round up to a multiple of 4)

Clients

The following clients access the data definition:

Physical Volume Repository, Storage System Manager, UniTree Migration.

6.2.5. Volume Structure - vol_t

Description

Unique identifier for a volume in an HPSS system.

Format

The Volume ID has the following format:

```
typedef struct {
    char    Vol[HPSS_PV_NAME_SIZE];
} vol_t;
```

Vol

ID of the volume. VOL_LABEL_SIZE is defined as 8. The first six characters of the volume ID is identical to the cartridge ID. The last two characters of the volume ID are (hex) digits representing the side number, where "00" is the first (or only) side.

HPSS_PV_NAME_SIZE is defined as 12, which is VOL_LABEL_SIZE + 1 (for a trailing NULL) + 3 (to round up to a multiple of 4).

Clients

The following clients access the data definition:

Storage Server, Storage System Manager.

6.2.6. Media Type Structure - media_type_t

Description

Identifies the media type of a cartridge or drive. For each media class, a bitmap describes all possible configurations (for example, cartridge size and compression mode). When a media type is applied to a drive, the bitmap shows all possible media types supported by the drive. When media type is applied to a cartridge, the bitmap shows the type of the cartridge. If a drive and a cartridge have the same media class, then a simple bitwise operation can be used to determine if the drive supports the cartridge. The operation looks like:

```
if ((drive.media.bitmap & cart.media.bitmap)==cart.media.bitmap)
    /* Drive supports cartridge */
else
```

```
/* Drive does not support cartridge */
```

Format

The Media Type has the following format:

```
typedef struct {
    media_class_t  Type;
    unsigned32     Subtype;
} media_type_t;
```

Type

A defined media type. One of:

```
MT_CLASS_TAPE
MT_CLASS_DISK
MT_CLASS_RWOPTICAL
MT_CLASS_WORMOPTICAL
MT_CLASS_MEMORY
```

Subtype

Different for each media type.

For Media Type MT_TAPE_DD2:

```
MT_SUB_25GB
MT_SUB_75GB
MT_SUB_165GB
```

For Media Type MT_TAPE_8MM:

```
MT_SUB_2_3GB
MT_SUB_5GB
```

Clients

The following clients access the data definition:

Storage Server, Storage System Manager.

6.2.7. Active Volume State Structure - activity_data_t

Description

Holds the current state of an active volume for a particular job. This is part of the larger Activity Structure. This part contains no pointers, is stored in metadata, and contains enough information to rebuild the pointers during PVL restart.

Format

The Activity Data Structure has the following format:

```
typedef struct {
    job_id_t      JobID;
    vol_t         Vol;
    drive_type_t  DriveType;
    signed32      DriveOption;
    signed32      DriveCount;
    drive_t       Drives[PVL_MAX_DRIVES_PER_PVL];
    drive_t       Drive;
    unsigned32    ActState;
    client_t      Client;
} activity_data_t;
```

JobID

Job ID.

Vol

Volume ID.

DriveType

Type of drive needed, from PhysicalVolumeType of the Volume.

DriveOption

One of the following:

PVL_ANY

PVL_INCLUDE

PVL_EXCLUDE

DriveCount

The number of entries in the Drives array.

Drives

Specific drive ids to be included or excluded, depending on DriveOption.

DriveAssigned

ID of drive assigned to volume.

ActState

State of activity. One of:

| | |
|-----------------------|--|
| PVL_ACT_UNCOMMITTED | The job which manages the volume has not yet been committed. |
| PVL_ACT_CART_WAIT | The cartridge containing the volume is not available (i.e. it is in use by another job). |
| PVL_ACT_CART_ASSIGNED | The cartridge has been reserved. This is a transitory state between the states PVL_ACT_CART_WAIT and PVL_ACT_DRIVE_WAIT. |
| PVL_ACT_DRIVE_WAIT | The cartridge containing the volume has been reserved, but no drives with the requested characteristics are available. |
| PVL_ACT_MOUNT_PENDING | The volume is in the process of being mounted in the drive. |
| PVL_ACT_MOUNT_FAILED | The volume was not mounted successfully. This is a transitory state. |
| PVL_ACT_MOUNTED | The volume is in the drive and available for read/write operations. |

| | |
|--------------------------|--|
| PVL_ACT_DISMOUNT_PENDING | The volume is in the process of being dismounted from the drive. |
| PVL_ACT_INJECT | The volume is in the process of being injected. |
| PVL_ACT_EJECT | The volume is in the process of being ejected.. |
| PVL_ACT_READING_LABEL | The volume label is in the process of being read after the PVR has notified the PVL that the cartridge has been mounted. |
| PVL_ACT_UNLOAD_PENDING | The volume is in the process of being elevated from the drive and dismounting. |
| PVL_ACT_DISMOUNTED | The volume is dismounted. |

Client

ID of the Client process which requested this volume.

Clients

The following clients access the data definition:

Storage Server, Storage System Manager.

6.2.8. Activity Structure - activity_t

Description

This is the PVL's full internal Activity structure. It contains the smaller activity data that can be stored in metadata, plus pointers to related activity, job, and active cart list structures.

Format

The Activity Structure has the following format:

```
typedef struct activity_s {
    activity_data_t    Act;
    struct activity_s *PrevJob;
    struct activity_s *NextJob;
    struct activity_s *PrevCart;
    struct activity_s *NextCart;
    struct job_ent_s  *ParentJob;
    struct cart_ent_s *ParentCart;
    client_info_t     *Client;
    drive_index_t     DriveIx;
    bool_t             DriveAllocated;
    bool_t             ThreadActive;
    drive_type_ent_t  *DriveTypeEnt;
    time_t             DelayTime;
    signed32           LabelFormat;
} activity_t;
```

Act

The smaller Activity Data structure (see activity_data_t).

PrevJob, NextJob

Pointers to activity structures in other jobs requesting the same cartridge.

PrevCart, NextCart

Pointers to other activity structures in the same job.

ParentJob

Pointer to the job containing this activity.

ParentCart

Pointer to the active cart list entry for this cartridge.

Client

ID of the client which requested this volume.

DriveIx

PVL's internal index for the drive which has this volume mounted, if any.

DriveAllocated

Boolean indicating whether the drive has been allocated from the PVL's drive count.

ThreadActive

Boolean indicating whether there is a thread currently managing this activity.

DriveTypeEnt

Pointer to the drive type information.

DelayTime

Time this activity was placed in deferred dismount state.

LabelFormat

Label format. One of the following:

PVL_LABEL_NONE

PVL_LABEL_FOREIGN

PVL_LABEL_HPSS

PVL_LABEL_DATA

PVL_LABEL_NON_ANSI

Clients

The following clients access the data definition:

None.

6.2.9. Client Information Structure - client_info_t

Description

Information to associate a client and the client's asynchronous notification with a particular job.

Format

The Client Info Structure has the following format:

```
typedef struct client_info_s {
```

```

    client_t          *Client;
    struct client_info_s *Next;
} client_info_t;

```

Client

Pointer to the Client Cache entry structure.

Next

Next client associated with the particular job.

Clients

The following clients access the data definition:

None.

6.2.10. Job Data Structure - job_data_t**Description**

Information about a single job (Mount, Import, Export, or Move) currently being managed by the PVL. . This is part of the larger Job Structure. This part contains no pointers, is stored in metadata, and contains enough information to rebuild the job list and drive queue pointers during PVL restart.

Format

The Job Data Entry has the following format:

```

typedef struct {
    job_id_t          JobID;
    u_signed64        RegisterBitmap;
    timestamp_t       Created;
    timestamp_t       CommitTime;
    unsigned32        Priority;
    unsigned32        JobState;
    unsigned32        JobType;
    unsigned32        ActCount;
    unsigned32        InDriveQueue;
    job_id_t          DriveQNext;
    signed32          ControllerWait;
} job_data_t;

```

JobID

Job ID. Generated by the PVL and used by the client and SSM to reference the job.

RegisterBitmap

Bitmap that shows which job attributes the SSM has registered for change notification.

Created

Time when job was first created.

CommitTime

Time when job was committed if the job is a mount.

Priority

Reserved for future use in scheduling jobs.

JobState

Current state of the job. One of:

| | |
|--------------------------|---|
| PVL_JOB_UNCOMMITTED | The job has not yet been committed. |
| PVL_JOB_CART_WAIT | One or more cartridges need by the job are not available (i.e. in use by another job). |
| PVL_JOB_DRIVE_WAIT | The cartridges containing the volumes have been reserved, but one or more drives with the requested characteristics are not available. |
| PVL_JOB_MOUNT_WAIT | The cartridges containing the volumes have been reserved, and the drives are available, but one or more requested volumes are in the process of being mounted. |
| PVL_JOB_MOUNTED | All cartridges are in their drives and the job is available for read/write operations. Any client with notifications registered for the job would be notified when the job state becomes MOUNTED. |
| PVL_JOB_DISMOUNT_PENDING | A dismount has been requested for the job. The cartridges are being dismounted. The cartridges and drives are being assigned to other jobs. Once all of the dismounts have completed, the job is removed from the job list. |
| PVL_JOB_ABORTING | One or more volumes encountered mount errors. Any cartridges which were successfully mounted will be dismounted. Once all of the dismounts have completed, the job is removed from the job list. |
| PVL_JOB_INJECT | The Import job is requesting the PVR to Inject the cartridge. |
| PVL_JOB_EJECT | The Export job is requesting the PVR to eject the cartridge. |
| PVL_JOB_INUSE | All volumes in job have been mounted. |
| PVL_JOB_DEFER_DISMOUNTS | This job will contain all volumes in deferred dismount state. |

JobType

Type of job. One of PVL_MOUNT, PVL_IMPORT_DEFAULT, PVL_IMPORT_SCRATCH, PVL_DEFER_DISMOUNTS, PVL_EXPORT, PVL_MOVE, PVL_RELABEL.

ActCount

The number of activity structures attached to this job (the number of volumes to be used).

InDriveQueue

Boolean indicating whether this job is in the Drive Queue.

DriveQNext

JobID of the next job in the drive queue.

ControllerWait

Boolean value, set to true if volumes to be mounted in the job should be placed on drives with separate controllers(not implemented in the current release).

Clients

The following clients access the data definition:

None.

6.2.11. Job Entry Structure - job_ent_t**Description**

Information about a single job (Mount, Import, Export, or Move) currently being managed by the PVL. . This is the larger Job Structure, containing the job data which is stored in metadata plus pointers.

Format

The Job List Entry has the following format:

```
typedef struct job_ent_s {
    job_data_t      Job;
    bool_t          ThreadWaiting;
    hpsssem_t       JobSem;
    client_info_t   *ClientInfo;
    struct activity_s *Activity;
    struct job_ent_s *Prev;
    struct job_ent_s *Next;
    struct job_ent_s *PrevD;
    struct job_ent_s *NextD;
    signed          DismountReason;
    bool_t          AtomicJob;
    int             MountedVolumes;
} job_ent_t;
```

Job

The smaller job data structure which is stored in metadata. (see job_data_t).

ThreadWaiting

Boolean indicating whether there is a thread actively managing this job.

JobSem

A semaphore used when the thread managing the job is waiting on external events.

ClientInfo

Pointer to the list of Client(s) which are part of this job.

Activity

Pointer to the list of Activities which are part of this job.

Prev, Next

Pointers to other jobs in the (unordered) job list, if any.

PrevD, NextD

Pointers to other jobs in the (ordered) DriveQueue, if any.

DismountReason

The index of the log message indicating the reason the job is being dismounted, if any.

AtomicJob

Boolean indicating whether this job requires atomic mount.

MountedVolumes

The number of volumes currently mounted.

Clients

The following clients access the data definition:

None.

6.2.12. Cartridge List Entry Structure - cart_ent_t

Description

Information about any cartridge which is currently in use or requested by one or more PVL jobs. This is an internal data structure not visible outside the PVL.

Format

The Cartridge List Entry has the following format:

```
typedef struct cart_ent_s {
    cart_t          Cart;
    pvr_index_t     PvrIx;
    struct cart_ent_s *Prev;
    struct cart_ent_s *Next;
    activity_t      *Activity;
} cart_ent_t;
```

Cart

Cartridge ID.

PvrIx

PVL's internal index into a data structure representing all the PVRs.

Activity

Pointer to the list of activity structures with volumes that are contained in this cartridge.

Prev, Next

Pointers to the previous and next active cartridge list entries, respectively.

Clients

The following clients access the data definition:

None.

6.2.13. Volume Data Structure - vol_data_t

Description

Information about a specific volume in HPSS. This information is stored in Metadata and is not cached in memory.

Format

The Volume Data Structure has the following format:

```
typedef struct {
    vol_t      PhysicalVolumeID;
    signed32   Version;
    u_signed64 RegisterBitmap;
    uuid_t     PVR;
    client_t   AllocatedClientID;
    signed32   AllocationStatus;
    media_type_t PhysicalVolumeType;
    signed32   VolumeLabelFormat;
    signed32   OperationalState;
    signed32   UsageState;
    signed32   AdministrativeState;
    cart_t     CartridgeID;
} vol_data_t;
```

PhysicalVolumeID

Volume ID.

Version

The number identifying the version of the PVL which understands this record.

RegisterBitmap

Bitmap that shows which volume attributes the SSM has registered for change notification.

PVR

The UUID of the PVR containing the volume.

AllocatedClientID

The UUID of the client which has allocated the volume, if any.

AllocationStatus

Either PVL_ALLOCATED or PVL_UNALLOCATED.

PhysicalVolumeType

The media type of the volume.

VolumeLabelFormat

<unused>.

OperationalState

Reserved for future use.

UsageState

Reserved for future use.

AdministrativeState

Reserved for future use.

CartridgeID

The ID of the cartridge containing the Volume. This is the first six characters of the VolumeID.

Clients

The following clients access the data definition:

Storage System Manager, NSL UniTree Migration.

6.2.14. Drive Data Structure - `drive_data_t`

Description

Information about a specific drive in HPSS. This information is stored in Metadata and is also cached in memory.

Format

The Drive Data Structure has the following format:

```
typedef struct {
    drive_t      DriveID;
    unsigned32   Version;
    u_signed64   RegisterBitmap;
    uuid_t       PVR;
    drive_addr_t DriveAddress;
    media_type_t DriveType;
    unsigned32   Controller;
    signed32     PollingInterval;
    timestamp_t  MaintenanceDate;
    vol_t        MountedVolume;
    unsigned32   MountsSinceLastMaint;
    unsigned32   OperationalState;
    unsigned32   UsageState;
    unsigned32   AdministrativeState;
    unsigned32   DriveState;
    uuid_t       MvrID;
} drive_data_t;
```

DriveID

Drive ID shared by PVL, PVR, and Mover. Each drive must have a unique ID.

Version

The number identifying the version of the PVL which understands this record.

RegisterBitmap

Bitmap which defines the fields that SSM has registered for change notification. This bitmap will be stored in metadata, so notifications will be persistent across PVL restarts.

PVR

The UUID of the PVR containing the volume.

PVR which holds the drive. Every drive in an HPSS system is managed by exactly one PVR.

ControllerID

ID of the hardware controller that is used to access the drive. This is used to attempt to mount a set of volumes to drives on different controllers.

DriveAddress

Character string to identify the drive, used by PVR and Robot.

DriveType

Type of drive. Used to determine the media types that are supported by the drive.

MaintenanceDate

Time the drive was last maintained.

MountedVolume

Volume currently mounted in drive.

MountsSinceLastMaint

Number of mounts since the drive was last maintained.

OperationalState

Either ST_ENABLED, ST_DISABLED or ST_SUSPECT.

Note: special case, if OperationalState is ZERO, it is reset to ST_ENABLED.

UsageState

Either ST_IDLE or ST_BUSY.

AdministrativeState

Reserved for future use.

DriveState

Current state of the drive. One of:

| | |
|----------------------------|---|
| PVL_DRIVE_FREE | The drive is currently not being used. |
| PVL_DRIVE_IN_USE | The drive is currently being used. |
| PVL_DRIVE_DISMOUNT_PENDING | The volume on the drive is being removed and a dismount is in progress. |

MvrID

UUID of the Mover that accesses the drive.

Clients

The following clients access the data definition:

Storage Server, Storage System Manager.

6.2.15. Drive Index - drive_index_t

Description

This structure is an index into the drive table maintained by the server.

Format

```
typedef unsigned32 drive_index_t;
```

Clients

The following clients access the data definition:

None.

6.2.16. Drive ID - drive_t

Description

This structure denotes the ID of a drive as specified in the PVL/Mover Device/Drive List metadata.

Format

```
typedef signed32 drive_t;
```

Clients

The following clients access the data definition:

Physical Volume Repository, Storage System Manager.

6.2.17. Drive Type - drive_type_t

Description

This structure specifies the media type of a drive.

Format

```
typedef media_type_t drive_type_t;
```

Clients

The following clients access the data definition:

None.

6.2.18. Drive Type Entry - drive_type_ent_t

Description

This structure is used by the PVL to track the current state and availability of managed drive types.

Format

```
typedef struct drive_type_ent_s {
    drive_type_t          DriveType;
    signed32              NumDrives;
    signed32              AvailDrives;
    signed32              DeferDrives;
    drive_index_t         DriveIx;
    struct drive_type_ent_s *Next;
} drive_type_ent_t;
```

DriveType

The type of drive to track information on.

NumDrives

The number of drives of this type managed by the PVL which are in working order.

AvailDrives

The number of drives idle of this type.

DeferDrives

The number of mounted (in deferred dismount state) available drives.

DriveIx

The index of the first drive of this type.

Next

Pointer to the next drive type in the PVL.

Clients

The following clients access the data definition:

None.

6.2.19. Job ID - job_id_t**Description**

This structure specifies the job id of the request.

Format

```
typedef signed32 job_id_t;
```

Clients

The following clients access the data definition:

Storage Server, Physical Volume Repository, Storage System Manager.

6.2.20. PVR Index - pvr_index_t**Description**

This structure specifies an index into a table of PVRs.

Format

```
typedef unsigned32 pvr_index_t;
```

Clients

The following clients access the data definition:

None.

6.2.21. Queue Data - queue_data_t**Description**

This structure is used internally by the PVL and is identical with the `api_queue_data_t` structure with the exception of the last field in the structure `Jobs`. In `api_queue_data_t` the `JobList` is implemented as a conformant array, in this structure it is implemented as a list via pointer. The two structures contain identical information.

Format

```
typedef struct {
    signed32      QueueID;
    signed32      Version;
    u_signed64    RegisterBitmap;
    signed32      TotalRequests;
    job_id_list_t *Jobs;
} queue_data_t;
```

QueueID

There is currently only one Queue. This field is for future use.

Version

The number identifying the version of the PVL which understands this record.

RegisterBitmap

Bitmap that show which job attributes the SSM has registered for change notification.

TotalRequests

The number of requests in the PVL's queue.

Jobs

Pointer to a job structure in the queue or NULL if the queue is empty.

Clients

The following clients access the data definition:

None.

6.3. Other Interfaces

6.3.1. `ss_MountCallback`

Purpose

Allows PVL to notify clients when mounts have completed.

Syntax

```
signed32 ss_MountCallback(
    handle_t      Bh,           /* IN */
    job_id_t     JobId,        /* IN */
    vol_t        Volume,       /* IN */
    drive_data_t DriveData,    /* IN */
    signed32     MountError); /* IN */
```

Description

This function should be provided by any client that calls `pvl_MountNew`. The function will be called by the PVL as each volume is mounted. The function may be called as soon as the `pvl_MountCommit` function is called (possibly before the `pvl_MountCommit` function returns). The function will be called as an RPC by the PVL.

Parameters

| | |
|-------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>JobId</i> | Job ID (returned by <code>pvl_MountNew</code>) of the job owning the mounted volume. |
| <i>Volume</i> | ID of the volume which was mounted. |
| <i>DriveData</i> | Information about the drive to which the volume was mounted. |
| <i>MountError</i> | Indicates the status of the mount. Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. On error <i>DriveData</i> is undefined. |

Error conditions

| | |
|---------------|------------------------------------|
| HPSS_ENOMOUNT | Mount failed. |
| HPSS_ENOENT | Couldn't find request for volume. |
| HPSS_EEXIST | Already notified about this mount. |

Return values

The return value is ignored.

Error conditions

None.

See also

`pvl_MountNew`, `pvl_MountAdd`, `pvl_MountCommit`.

Clients

Storage Server will provide the RPC interface for this function, the Physical Volume Library is a client. The function is included in `libpvl.c`, which provides a simplified API to the Storage Server.

7. Physical Volume Repository Functions

This chapter specifies the Physical Volume Repository programming interface. Specifically, the following information is provided:

Application Programming Interfaces (APIs)

Data Definitions

7.1. API Functions

This section describes all APIs which are provided for use by another HPSS subsystem or by a client external to HPSS. The API interface specification includes the following information:

Name

Syntax

Description

Parameters

Return Values

Error Conditions

Related Information

Clients

Notes

7.1.1. pvr_Audit

Purpose

Audit all or part of a repository checking external labels on cartridges when possible. Not implemented in the current release.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_Audit(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    location_t        *Location); /* IN */
```

Description

An area in the repository is audited to compare the locations of the actual cartridges with their locations as stored in metadata. Every location in a repository is identified by four labels -- unit, panel, row, column. This function should be used to reset the internal state of a repository after any cartridges have been manually moved.

The function returns immediately and continues to run asynchronously. Appropriate status messages will be sent to Storage System Manager indicating progress of the audit.

Parameters

| | |
|-----------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Location</i> | Location(s) to audit. If any of the location components (unit, panel, row, column) are set to -1, all of those locations will be audited. For example, to audit all of unit 5, panel 3, the values of location should be {5, 3, -1, -1}. This argument may be ignored by the repository. All location arguments after the first -1 are assumed to be -1. Therefore all of unit 5, panel 3 will be audited if the location is {5, 3, -1, 2}. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. The audit itself continues to run asynchronously after the function returns.

Error conditions

| | |
|--------------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_EBUSY | Another audit is already in progress. |
| HPSS_ENOTSUPPORTED | Not implemented. |

See also

[pvr_QueryCartridge](#), [pvr_SetCartridge](#).

Clients

Storage System Manager.

Notes

This API is not implemented in the current release. The PVL is notified via the **pvl_NotifyCartridge** API about each cartridge that is found. This helps keep the PVL and PVR metadata in synchronization.

7.1.2. pvr_CartridgeGetAttrs

Purpose

Get the current values of the attributes of a cartridge.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_CartridgeGetAttrs(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t Ch,      /* IN */  
    cart_t            Cartridge,   /* IN */  
    cart_data_t       Attributes); /* OUT */
```

Description

Returns the value of all Cartridge attributes for the cartridge specified by the Cartridge argument.

Parameters

| | |
|-------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Cartridge</i> | ID of cartridge managed object to retrieve. |
| <i>Attributes</i> | Structure containing all attributes of the Cartridge Managed Object. Note that the definition of this structure is maintained in the SSM chapter. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of Attributes is undefined.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOENT | Cartridge not found. |
| HPSS_EINVAL | Invalid input argument. |

See also

pvr_CartridgeSetAttrs.

Clients

Storage System Manager.

Notes

None.

7.1.3. `pvr_CartridgeSetAttrs`**Purpose**

Set the current values of the attributes of a cartridge.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_CartridgeSetAttrs(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    cart_t            *Cartridge,   /* IN */
    u_signed64        InSelectBitmap, /* IN */
    u_signed64        *OutSelectBitmap, /* OUT */
    cart_data_t       *InAttributes, /* IN */
    cart_data_t       *OutAttributes); /* OUT */
```

Description

Sets cartridge attributes for the cartridge specified by the *Cartridge* argument to the value of the corresponding attributes of the *InAttributes* argument. Only those attributes identified by the *InSelectBitmap* field are set.

Parameters

| | |
|------------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Cartridge</i> | ID of managed object to set. |
| <i>InSelectBitmap</i> | Indicates which object attributes are to be set. |
| <i>OutSelectBitmap</i> | Indicates which object attributes were actually set. |
| <i>InAttributes</i> | New values of attributes to be set. |
| | The <i>RegisterBitmap</i> field may be set to any value. Note that this is the only field which may be set when updating the Generic Cartridge Object. |
| | The <i>MaintenanceDate</i> field may be set to any value. |
| | The <i>MountsSinceMaint</i> field may be set to any value. |
| <i>OutAttributes</i> | Entire managed object, including newly updated fields. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EINVAL | Attempted to set a read-only attribute. |

HPSS_EMDM Metadata manager failure.

HPSS_ENOENT *Cartridge* not found.

See also

pvr_CartridgeGetAttrs.

Clients

Storage System Manager.

Notes

If the *RegisterBitmap* field is set, it will be stored in metadata and remain set across PVR restarts.

If the *CartridgeID* field of the attributes structure contains the ID of the generic cartridge object, then only the *RegisterBitmap* can be set. This bitmap will be ORed with the *RegisterBitmap* attribute of each cartridge object to determine if a notification is to be sent to the SSM.

7.1.4. `pvr_CheckIn`**Purpose**

Notify operator to check in a previously checked out cartridge.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_CheckIn(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    cart_data_t      *CartData,  /* IN */
    location_t       *Location); /* IN */
```

Description

The PVR uses its check in mechanism to accept a previously checked out cartridge. The external label on the cartridge is verified if possible.

Parameters

| | |
|-----------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>CartData</i> | Metadata describing the cartridge. |
| <i>Location</i> | Location where new cartridge was inserted; may be NULL if the standard inject port on the repository was used. |

Return Values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error Conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EINVAL | The cartridge was not specified. |
| HPSS_ENOENT | Cartridge not found. |
| HPSS_EMDM | Metadata manager failure. |

See Also

`pvr_CheckOut`.

Clients

PVL.

Notes

None.

7.1.5. pvr_CheckOut

Purpose

Check out a cartridge from a PVR.

Syntax

```
#include "pvr_interface.h"

signed32 pvr_CheckOut(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    cart_t           *Cart,       /* IN */
    location_t       *Location,   /* IN */
    cart_data_t      *CartData); /* OUT */
```

Description

The PVR ejects the cartridge to the operator. All metadata associated with the cartridge is returned to the client.

Parameters

| | |
|-----------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Cart</i> | Cartridge to check out. |
| <i>Location</i> | Location to place the cartridge when checking out or location currently holding cartridge, may be NULL if the standard eject port on the repository is used. |
| <i>CartData</i> | Pointer to client provided space to return information about the checked out cartridge. |

Return Values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error code is returned, the contents of *CartData* are undefined.

Error Conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EINVAL | The cartridge was not specified. |
| HPSS_EBUSY | Cartridge is currently mounted. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOENT | Cartridge not found. |

See Also

pvr_CheckIn.

Clients

PVL.

Notes

None.

7.1.6. pvr_DismountCart

Purpose

Dismount a single cartridge.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_DismountCart(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    cart_t           *Cart);     /* IN */
```

Description

Dismounts the cartridge if it is currently in a drive and returns it to a storage location.

This function runs synchronously. That is, it doesn't return to the caller until the cartridge is dismounted. For some repositories, the dismount is only performed logically, the cartridge might be left in the drive until the drive is needed by another cartridge.

Parameters

| | |
|-------------|-----------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Cart</i> | Cartridge to be dismounted. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EINVAL | Cartridge was not specified. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOENT | Cartridge not found. |

See also

pvr_Mount.

Clients

Physical Volume Library.

Notes

None.

7.1.7. `pvr_DismountDrive`**Purpose**

Dismounts the cartridge in a given drive.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_DismountDrive(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    drive_t          Drive);    /* IN */
```

Description

Dismounts the given drive and returns any mounted cartridge to its storage location.

This function should be used only by the Storage System Manager to force a dismount when `pvr_DismountCart` fails. This function will attempt to dismount a drive even if it believes that no cartridge is currently in the drive.

Parameters

| | |
|--------------|----------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Drive</i> | Drive to dismount. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EINVAL | The drive was not specified. |
| HPSS_ENODEV | The drive does not exist. |
| HPSS_ENOENT | The drive could not be dismounted. |
| HPSS_ENOTSUP | Operation not supported by current version of the PVR. |

See also

`pvr_Mount`, `pvr_DismountCart`.

Clients

Physical Volume Library.

Notes

None.

7.1.8. pvr_Eject

Purpose

Ejects a cartridge from a PVR.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_Eject(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t Ch,      /* IN */  
    cart_t            Cart,        /* IN */  
    location_t        Location,    /* IN */  
    cart_data_t       CartData);   /* OUT */
```

Description

The PVR ejects the cartridge to the operator. All metadata associated with the cartridge is returned to the client.

Parameters

| | |
|-----------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Cart</i> | Cartridge to eject. |
| <i>Location</i> | Location to place the cartridge when ejecting or Location currently holding cartridge, may be NULL if the standard eject port on the repository is used. |
| <i>CartData</i> | Pointer to client provided space to return information about the ejected cartridge. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error code is returned, the contents of *CartData* are undefined.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EINVAL | The cartridge was not specified. |
| HPSS_EBUSY | Cartridge is currently mounted. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOENT | Cartridge not found. |

See also

pvr_Inject.

Clients

Physical Volume Library.

Notes

None.

7.1.9. pvr_Inject

Purpose

Accept a new cartridge in the PVR.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_Inject(  
    handle_t                Bh,           /* IN */  
    hpss_connect_handle_t  *Ch,           /* IN */  
    cart_data_t            *CartData,     /* IN */  
    location_t             *Location);    /* IN */
```

Description

The PVR uses its inject mechanism to accept a new cartridge. The external label on the cartridge is verified if possible.

Parameters

| | |
|-----------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>CartData</i> | Metadata describing new cartridge, may be NULL and will be ignored if the cartridge already exists in metadata. |
| <i>Location</i> | Location where new cartridge was inserted, may be NULL if the standard inject port on the repository was used. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EEXIST | A cartridge with the same ID is already in the repository. |
| HPSS_EINVAL | Number of sides not between 1 and 256 inclusive. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOSPACE | The repository is full. |

See also

pvl_Eject.

Clients

Physical Volume Library.

Notes

None.

7.1.10. pvr_ListAllCart

Purpose

List all cartridges managed by the PVR.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_ListAllCart(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    signed32          Continue,   /* IN */
    cart_data_t       *Buffer,    /* OUT */
    unsigned32        BufferEntries, /* IN */
    unsigned32        *NumEntries); /* OUT */
```

Description

Lists all cartridges managed by the PVR. If *Buffer* is not large enough to hold all of the cartridges, the function should be called again with *Continue* set to true. This should be done until the value returned in *NumEntries* is less than the number of cartridges in *Buffer*.

Parameters

| | |
|----------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Continue</i> | If FALSE, perform a new selection. Otherwise, if TRUE, continue previous call to this function. |
| <i>Buffer</i> | Pointer to client allocated array of cartridge data structures. Cartridge data will be returned in this array. |
| <i>BufferEntries</i> | Number of entries in buffer. |
| <i>NumEntries</i> | The number of cartridge data structures placed in <i>Buffer</i> . |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EINVAL | Invalid argument. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_EAGAIN | Successful read, there is still cartridge information left unread. Call the function again with <i>Continue</i> set. |

See also

pvr_QueryCartridge.

Clients

Storage System Manager.

Notes

This function is reentrant, but it should not be called simultaneously by two or more non-cooperating clients. The value of the *Continue* argument refers to the last function call made, not the last call made by the specific client.

7.1.11. `pvr_ListPendingMounts`**Purpose**

List all currently pending mounts for the PVR.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_ListPendingMounts(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    pending_mount_t  *Buffer,     /* OUT */
    unsigned32       BufferEntries, /* IN */
    unsigned32       *NumEntries); /* OUT */
```

Description

Lists all currently pending mounts for the PVR. Since robotics mounts complete quickly, this list should be small or empty for robotics repositories. The list may be longer for operator mounted drives. In any case, the list should never be longer than the number of drives in the PVR because the PVR only accepts a mount request for a drive that is currently empty.

If *Buffer* is not large enough to hold all of the cartridges, the function should be called again with a larger *Buffer*. There is no *Continue* flag like in `pvr_ListAllCart`.

Parameters

| | |
|----------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Buffer</i> | Client allocated array of entries to place mount information when returning. |
| <i>BufferEntries</i> | Number of entries in <i>Buffer</i> array. Note: This is the number of entries in the array, not the size of the array in bytes. |
| <i>NumEntries</i> | Number of entries returned in the array. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EINVAL | Invalid argument. |
| HPSS_ECONN | Not initialized. |
| HPSS_EAGAIN | <i>Buffer</i> too small. Check <i>NumEntries</i> and try again with a buffer large enough to hold all entries. |

See also

`pvr_CartridgeGetAttrs`.

Clients

Storage System Manager.

Notes

None.

7.1.12. `pvr_Mount`**Purpose**

Asynchronously mount a single cartridge.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_Mount(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    cart_t            *Cart,        /* IN */
    side_t            Side,         /* IN */
    signed32          DriveOption,  /* IN */
    signed32          DriveCount,   /* IN */
    drive_t           *DriveList[], /* IN */
    job_id_t          JobId);      /* IN */
```

Description

Mounts the given cartridge/side in a drive. Depending on the drive selection option input. The drive may be specified in the input drive list or any drive available to the PVR.

This function returns immediately and continues to attempt to mount the cartridge asynchronously. When the mount completes or fails, the function calls **pvl_MountComplete** to notify the PVL. It is also expected that the PVL will periodically poll the drives so that the PVL can determine when the cartridge is mounted. The PVR will not always know when the cartridge is mounted and therefore will not be able to call **pvl_MountComplete**. The PVL should call **pvr_MountComplete** when it determines that a cartridge has been mounted.

Parameters

| | |
|--------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Cart</i> | Cartridge to be mounted. |
| <i>Side</i> | Side of cartridge to be mounted. |
| <i>DriveOption</i> | Select drive from drive list or any drive in PVR. |
| <i>DriveCount</i> | Length of <i>DriveList</i> . |
| <i>DriveList</i> | List of possible drives for mount. The PVR may pick any drive from this list. |
| <i>JobId</i> | Unique ID generated by PVL which is associated with a set of mounts. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

A successful return code indicates that the mount is in progress. The client can poll the device to determine when the cartridge is available, or wait for the PVR to call **pvl_MountComplete**.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EAGAIN | The mount interface (e.g. robot) is not available. |
| HPSS_EBUSY | Cartridge is already mounted. |
| HPSS_EINVAL | Nonexistent cartridge side specified. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENODEV | No drive available for mount. |
| HPSS_ENOENT | Cartridge not found. |
| HPSS_EOUT | The cartridge is checked out of the PVR. |

See also

pvr_MountComplete, **pvr_DismountCart**.

Clients.

Physical Volume Library.

Notes

The client should be aware that **pvl_MountComplete** could be called before the call to **pvr_Mount** returns.

7.1.13. pvr_MountComplete**Purpose**

Client is notifying the PVR that a requested mount has completed.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_MountComplete(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    cart_t           *Cart,      /* IN */
    side_t           Side,        /* IN */
    drive_t          *Drive,     /* IN */
    signed32         Result);    /* IN */
```

Description

This function is called by the PVL when it detects a mounted cartridge that the PVR has not informed the PVL about. This function is usually required when an operator mounts a cartridge, because the PVR has no way to know when that mount has completed. The purpose of this function is to tell the PVR which drive the cartridge was mounted in and to tell the PVR to stop prompting for the cartridge mount (if necessary).

Parameters

| | |
|---------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Cart</i> | Cartridge that was mounted. |
| <i>Side</i> | Side of cartridge that was mounted. |
| <i>Drive</i> | Drive in which the cartridge was mounted. |
| <i>Result</i> | The result of the mount. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EINVAL | Nonexistent cartridge side specified. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_ENOENT | Cartridge not found. |

See also

[pvr_Mount](#), [pvr_DismountCart](#), [pvl_MountComplete](#).

Clients

Physical Volume Library.

Notes

None.

7.1.14. pvr_PVRGetAttrs**Purpose**

Get the current values of the attributes of the PVR.

Syntax

```
#include "pvr_interface.h"

signed32 pvr_PVRGetAttrs(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    pvr_data_t       *Attributes); /* OUT */
```

Description

Returns the value of all PVR attributes.

Parameters

| | |
|-------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Attributes</i> | Structure containing all attributes of the PVR Managed Object. Note that the definition of this structure is maintained in the SSM chapter. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of *Attributes* is undefined.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |

See also

pvr_PVRSetAttrs.

Clients

Storage System Manager.

Notes

None.

7.1.15. pvr_PVRSetAttrs

Purpose

Set the current values of the attributes of the PVR.

Syntax

```
#include "pvr_interface.h"

signed32 pvr_PVRSetAttrs(
    handle_t                Bh,                /* IN */
    hpss_connect_handle_t  *Ch,                /* IN */
    u_signed64              InSelectBitmap,     /* IN */
    u_signed64              *OutSelectBitmap,   /* OUT */
    pvr_data_t              *InAttributes,     /* IN */
    pvr_data_t              *OutAttributes);   /* OUT */
```

Description

Sets PVR attributes to the value of the corresponding field of the *InAttributes* argument. Only those attributes identified by the *InSelectBitmap* field are set.

Parameters

| | |
|------------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>InSelectBitmap</i> | Indicates which object attributes are to be set. |
| <i>OutSelectBitmap</i> | Indicates which object attributes were actually set. |
| <i>InAttributes</i> | New values of attributes to be set. The <i>RegisterBitmap</i> field may be set to any value. The <i>CartsAlarmThreshold</i> field may be set to any value. |
| <i>OutAttributes</i> | Entire managed object, including newly updated fields. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EMDM | Metadata manager failure. |
| HPSS_EINVAL | Attempted to set a read-only attribute. |

See also

pvr_PVRGetAttrs.

Clients

Storage System Manager.

Notes

The *RegisterBitmap* field will not be persistent across PVR restarts.

7.1.16. pvr_ServerGetAttrs

Purpose

Get the current values of the attributes of the PVR server.

Syntax

```
#include "pvr_interface.h"

signed32 pvr_ServerGetAttrs(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    hpss_server_attr_t *Attributes); /* OUT */
```

Description

Returns the value of all PVR server attributes.

Parameters

| | |
|-------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Attributes</i> | Structure containing all attributes of the HPSS Server Managed Object. Note that the definition of this structure is maintained in the SSM chapter. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned. If an error is returned, the value of *Attributes* is undefined.

Error conditions

| | |
|--------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |

See also

pvr_ServerSetAttrs.

Clients

Storage System Manager.

Notes

None.

7.1.17. `pvr_ServerSetAttrs`**Purpose**

Set the current values of the attributes of the PVR server.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 pvr_ServerSetAttrs(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    u_signed64         InSelectBitmap, /* IN */
    u_signed64         OutSelectBitmap, /* OUT */
    hpss_server_attr_t *InAttributes, /* IN */
    hpss_server_attr_t *OutAttributes); /* OUT */
```

Description

Sets PVR server attributes to the value of the corresponding field of the *InAttributes* argument. Only those attributes identified by the *InSelectBitmap* field are set.

Parameters

| | |
|------------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>InSelectBitmap</i> | Indicates which object attributes are to be set. |
| <i>OutSelectBitmap</i> | Indicates which object attributes were actually set. |
| <i>InAttributes</i> | New values of attributes to be set. |

The *RegisterBitmap* attribute may be set to any value.

The *AdministrativeState* field may be set to:

| | |
|-------------------|---|
| ST_HALT - | Halt the PVR immediately. |
| ST_SHUTDOWN - | Halt the PVR gracefully. |
| ST_REINITIALIZE - | Re-read the drive configuration information. |
| ST_REPAIRED - | Set <i>HardwareStatus</i> to STAT_ENABLED and set <i>OperationalState</i> to ST_NORMAL. |

The *HardwareStatus* field may be set to:

| | |
|-------------------|--|
| STAT_ENABLED - | also set <i>OperationalState</i> to ST_NORMAL. |
| Any other value - | also set <i>OperationalState</i> to ST_BROKEN. |

OutAttributes Entire managed object, including newly updated fields.

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------------|--|
| HPSS_ENOTYET | The server has not completed initializing. |
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_EINVAL | Attempted to set one or more read-only attributes. The <i>NotifyBitmap</i> indicates which attributes were successfully set. |
| HPSS_ENOTSUPPORTED | Reinitializing <i>AdministrativeState</i> is not supported for the PVR. |

See also

pvr_ServerGetAttrs.

Clients

Storage System Manager.

Notes

The *RegisterBitmap* field will not be persistent across PVR restarts.

7.2. Device Interfaces

7.2.1. **device_Audit**

Purpose

Reports the identity of cartridges in one or more storage locations. Not implemented for the current release.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 device_Audit(  
    location_t      *Location);    /* IN */
```

Description

This function starts a thread which runs asynchronously checking the cartridges in one or more locations. The thread calls the CartridgeMoved function in the generic PVR for every cartridge it finds (even those which don't appear to have been moved).

Parameters

| | |
|-----------------|---|
| <i>Location</i> | Identifies the location(s) to audit. A slot identifier has four fields - unit, panel, row, column. A value of -1 in one or more of these fields acts as a wild card meaning search all. For example, {5,10,-1,-1} will search all rows and columns in unit 5, panel 10. |
|-----------------|---|

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|--------------------|--------------------------|
| HPSS_ENOTSUPPORTED | Operation not supported. |
| HPSS_EOPNOTSUPP | Operation not supported. |

See also

pvr_Audit.

Clients

Physical Volume Repository.

Notes

Not implemented for the current release.

7.2.2. `device_Dismount`

Purpose

Issue device specific commands to dismount a cartridge.

Syntax

```
#include "pvr_interface.h"

signed32 device_Dismount(
    cart_t      *Cart,          /* IN */
    side_t      *Side,         /* IN */
    drive_addr_t *Drive,       /* IN */
    cart_t      *EjectedCart,  /* OUT */
    location_t  *Location);    /* OUT */
```

Description

This function will dismount the cartridge or the drive. If either is NULL, the other will be used to determine what to dismount. The caller should provide both arguments when possible. Depending on robot, a cartridge which is mounted in a different drive may result in an error being returned.

The function is responsible for verifying the external label on the cartridge if possible.

Parameters

| | |
|--------------------|--|
| <i>Cart</i> | Cartridge to dismount. May be NULL if drive is specified. |
| <i>Side</i> | Side of cartridge to dismount. Must be specified if Cart is specified. |
| <i>Drive</i> | Drive to dismount. May be NULL if cartridge is specified. |
| <i>EjectedCart</i> | Cartridge which was dismounted.. |
| <i>Location</i> | Identifies the slot that the cartridge was returned to after being dismounted. If the location argument is NULL, nothing will be returned. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|---|
| HPSS_ENODEV | Drive not found. |
| HPSS_ENOENT | Cartridge not found. |
| HPSS_EINVAL | Drive specified doesn't have a cartridge mounted. |

See also

`pvr_DismountCart`, `pvr_DismountDrive`.

Clients

Physical Volume Repository.

Notes

This function should succeed even if called before the original mount completes. It is acceptable if the function blocks until the mount completes and then performs the dismount.

7.2.3. device_Eject

Purpose

Issue device specific commands to eject a cartridge.

Syntax

```
#include "pvr_interface.h"

signed32 device_Eject(
    cart_data_t *CartData,    /* IN */
    location_t *Location);   /* IN */
```

Description

This function will eject a cartridge from a repository.

The function is responsible for verifying the external label on the cartridge if possible.

Parameters

| | |
|-----------------|--|
| <i>CartData</i> | Cartridge to eject. |
| <i>Location</i> | Location currently holding cartridge, or location to place cartridge when ejecting (device dependent). |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|--|
| HPSS_EBUSY | Unable to eject cartridge because the cartridge or the eject mechanism are in use. |
| HPSS_ENOENT | Cartridge not found. |

See also

pvr_Eject.

Clients

Physical Volume Repository.

Notes

None.

7.2.4. **device_Init**

Purpose

Allow the repository to perform any initialization necessary.

Syntax

```
#include "pvr_interface.h"
```

```
void device_Init(void)
```

Description

Called once when the PVR starts to allow any device specific initialization to be performed.

Parameters

None.

Return values

None.

Error conditions

None.

See also

device_Release.

Clients

Physical Volume Repository.

Notes

None.

7.2.5. device_Inject

Purpose

Issue device specific commands to inject a new cartridge.

Syntax

```
#include "pvr_interface.h"
```

```
signed32 device_Inject(  
    cart_data_t *CartData,          /* IN */  
    location_t *Location,          /* IN */  
    location_t *ActualLocation);   /* OUT */
```

Description

This function will accept a new cartridge in a repository. The function is responsible for verifying the external label on the cartridge if possible.

Parameters

| | |
|-----------------------|---|
| <i>CartData</i> | Data describing new cartridge. |
| <i>Location</i> | Location where the new cartridge was inserted. Not applicable for all robots. |
| <i>ActualLocation</i> | Identifies the slot the cartridge was placed in after being injected into the repository. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|---------------|--|
| HPSS_EEXIST | A cartridge with the same ID is already in the repository. |
| HPSS_ENOSPACE | The repository is full. |

See also

pvr_Inject.

Clients

Physical Volume Repository.

Notes

None.

7.2.7. device_Mount

Purpose

Issue the device specific commands necessary to mount a cartridge.

Syntax

```
#include "pvr_interface.h"

signed32 device_Mount(
    cart_t      *Cart,          /* IN */
    side_t      *Side,         /* IN */
    side_t      TotalSides,    /* IN */
    signed32    Count,         /* IN */
    drive_t     DriveList[],   /* IN */
    job_id_t    Job,          /* IN */
    uuid_t      *Uuid,        /* IN */
    short       *Tries);      /* IN */
```

Description

Commands are issued to the device to mount the requested cartridge in one of the drives specified in the drive list.

The function returns immediately and the mount continues in a separate thread. When the mount completes, the thread calls **pvl_MountCompleted** to notify the PVL that the cartridge is mounted. It also updates the *Status* and *CurrentLocation* attributes of the cartridge in metadata.

The thread is responsible for verifying the external label on the cartridge if possible.

Parameters

| | |
|-------------------|---|
| <i>Cart</i> | Cartridge to be mounted. |
| <i>Side</i> | Side of cartridge to be mounted. |
| <i>TotalSides</i> | Number of sides on the cartridge. |
| <i>Count</i> | Number of drives listed in <i>Drive</i> argument. |
| <i>DriveList</i> | List of possible drives for mount. The PVR may pick any drive from this list. The drive selected by the repository must be on-line. |
| <i>Job</i> | The ID of the job which this mount is a part of. |
| <i>Uuid</i> | ID of client that requested the mount. Used to send an asynchronous reply. |
| <i>Tries</i> | The number of times the mount has been attempted. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|---|
| HPSS_EAGAIN | The mount interface (e.g. robot) is not available. Error starting mount thread. |
|-------------|---|

HPSS_ENOMEM

Unable to allocate memory(malloc).

See also

pvr_Mount, device_Dismount.

Clients

Physical Volume Repository.

Notes

The client should be aware that **pvl_MountCompleted** could be called before the call to **device_Mount** returns.

7.2.8. **device_MountComplete**

Purpose

Allow any device specific cleanup when the client notifies the PVR that a cartridge is mounted.

Syntax

```
#include "pvr_interface.h"

signed32 device_MountComplete(
    cart_t      *Cart,      /* IN */
    side_t      *Side,     /* IN */
    drive_addr_t *DriveAddr); /* IN */
```

Description

This function is called when the **pvr_MountComplete** API is called. This API will be called by the client when the client detects that a cartridge has been mounted. The client is only required to call this API when it has not been notified (via **pvl_MountCompleted**) that the mount has completed. However, the client may call this API every time it determines that a cartridge has been mounted.

Parameters

| | |
|------------------|---|
| <i>Cart</i> | Cartridge that was mounted. |
| <i>Side</i> | Side of cartridge that was mounted. |
| <i>DriveAddr</i> | Drive in which the cartridge was mounted. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

None.

See also

pvr_MountComplete.

Clients

Physical Volume Repository.

Notes

None.

7.2.9. **device_Release**

Purpose

Allow the repository to perform any shutdown necessary.

Syntax

```
#include "pvr_interface.h"
```

```
void device_Release(void)
```

Description

Called once when the PVR shuts down to allow any device specific shut down to be performed.

Parameters

None.

Return values

None.

Error conditions

None.

See also

device_Init.

Clients

Physical Volume Repository.

Notes

It is not guaranteed that the PVR will call this function when shutting down. This function should return quickly.

7.2.10. `device_SetDrive`

Purpose

Places a drive on-line or off-line. Not implemented in the current release.

Syntax

```
#include "pvr_interface.h"

signed32 device_SetDrive(
    drive__t      *Drive,      /* IN */
    signed32      *Status);   /* IN */
```

Description

Used to place a drive on-line or off-line. A PVR should not try to mount to drives which are placed off-line.

Parameters

| | |
|---------------|-------------------------------------|
| <i>Drive</i> | Drive to change. |
| <i>Status</i> | One of PVR_ON_LINE or PVR_OFF_LINE. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------|-----------------------|
| HPSS_ENODEV | Drive does not exist. |
|-------------|-----------------------|

See also

`device_Mount`.

Clients

Physical Volume Repository.

Notes

If a drive is placed off-line while a cartridge is mounted, the drive will not go off-line until after the cartridge is dismounted.

7.3. Data Definitions

This section describes key internal data definitions and all externally used data definitions which are provided by this subsystem. A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

NOTE: We need to determine the total number of bytes of metadata required to scale the PVR to 500,000 physical volumes.

7.3.1. Cartridge Side - side_t

Description

Identifies the side of a cartridge. Side is a logical identifier which may refer to physical sides on an optical platter, or logical sides in multiple tape partitions.

Format

The Side ID has the following format:

```
typedef signed32 side_t;
```

Clients

The following clients access the data definition:

Physical Volume Library, Storage System Manager.

7.3.2. drive_addr_t

Description

String with name of drive device, for example, "/dev/rmt01".

Format

The Drive Address has the following format:

```
typedef struct {  
    char Drive[DRIVE_ADDR_SIZE];  
} drive_addr_t;
```

DRIVE_ADDR_SIZE is currently defined as 64.

Clients

The following clients access the data definition:

Storage System Manager.

7.3.3. ioport_addr_t

Description

String with name of I/O port.

Format

The I/O Port Address has the following format:

```
typedef struct {  
    char IOPort[IOPORT_ADDR_SIZE];  
} ioport_addr_t;
```

IOPORT_ADDR_SIZE is currently defined as 64.

Clients

The following clients access the data definition:

Storage System Manager.

7.3.3. location_t

Description

Identifies a physical location within a PVR. The physical location can be occupied by a cartridge. Ideally the port, slot, and drive fields could be combined in a union. However, union types are not allowed in HPSS data that will be stored by the metadata manager.

Format

The Location ID has the following format:

```
typedef struct {
    signed32  LocationType;
    port_t    Port;
    slot_t    Slot;
    drive_t   Drive;
} location_t;
```

LocationType

Identifies the type of the location. One of:

LOC_NONE

LOC_PORT

LOC_SLOT

LOC_DRIVE

Port

An import/export location or a passthru location (used to automatically pass a cartridge from one repository to another).

Slot

A location where cartridges are stored in a repository when they are not in use. Identified with four signed32 integers: unit, panel, row and column.

Drive

The unique ID of a drive in the HPSS system.

Clients

The following clients access the data definition:

Physical Volume Library, Storage System Manager.

7.3.4. cart_data_t

Description

Contains all metadata for a single cartridge.

Format

The Cartridge Data has the following format:

```
typedef struct {
    cart_t          Cart;
    signed32        Version;
    u_signed64      RegisterBitmap;
    side_t          CartSides;
    side_t          CartMountSide;
    uuid_t          PVRId;
    location_t      CartLocation;
    location_t      CartHomeLocation;
    media_type_t    CartType;
    manufacturer_t  ManufacturerID;
    lot_number_t    LotNumber;
    timestamp_t     ServiceStartDate;
    timestamp_t     MaintenanceDate;
    timestamp_t     LastMountedDate;
    signed32        MountsInService;
    signed32        MountsSinceMaint;
    signed32        OperationalState;
    signed32        UsageState;
    signed32        AdministrativeState;
    signed32        MountStatus;
    u_signed64      SecurityLevel[2];
    cos_t           ClassOfService;
    job_id_t        JobId;
} cart_data_t;
```

Cart

Cartridge ID. A six character string that represents the cartridge ID. This ID should be identical to the external label on the cartridge if one exists.

Version

Version number.

RegisterBitmap

Identifies those fields for which the SSM has registered to be notified when they change.

CartSides

Number of sides on the cartridges. Must be between 1 and 256 inclusive.

CartMountSide

The currently mounted side of the cartridge.

PVRId

ID of PVR managing the cartridge.

CartLocation

Current location of the cartridge.

CartHomeLocation

The home location of the cartridge if one exists.

CartType

Media type of the cartridge, i.e. 3480, D2, 8MM, etc. Refer to the *Physical Volume Library Functions* chapter for a description of `media_type_t`.

ManufacturerID

Cartridge manufacturer. Site defined string.

LotNumber

Cartridge lot number. Site defined string.

ServiceStartDate

Date cartridge entered HPSS system.

MaintenanceDate

Date cartridge was last cleaned or retensioned.

LastMountedDate

Date cartridge was last mounted.

MountsInService

Total number of mounts since cartridge entered HPSS system.

MountsSinceMaint

Number of mounts since the cartridge was last cleaned or retensioned.

OperationalState

SSM defined operational state.

UsageState

SSM defined usage state.

AdministrativeState

SSM defined administrative state.

MountStatus

One of: `PVR_MOUNT_PENDING`, `PVR_MOUNTED`, `PVR_DISMOUNT_PENDING`, `PVR_EJECT_PENDING` or `PVR_DISMOUNTED`.

SecurityLevel

Security Level of cartridge.

ClassOfService

HPSS class of service as applied to cartridge.

JobId

A mounted cartridge is a member of this job ID.

Clients

The following clients access the data definition:

Storage System Manager, NSL UniTree Migration.

7.3.5. pvr_data_t**Description**

Contains all metadata for a single PVR.

Format

The PVR Data has the following format:

```
typedef struct {
    uuid_t      PVRId;
    uuid_t      PVLId;
    signed32    Version;
    u_signed64  RegisterBitmap;
    signed32    TotalCartridges;
    signed32    CartsAlarmThreshold;
    signed32    CartsCapacity;
    signed32    SameJobOnController;
    signed32    OtherJobOnController;
    signed32    DistanceToDrive;
    u_signed64  CharacteristicsBitmap;
    char        CartFileName[HPSS_MAX_DCE_NAME];
    char        DeviceSpecificA[HPSS_MAX_DCE_NAME];
    char        DeviceSpecificB[HPSS_MAX_DCE_NAME];
} pvr_data_t;
```

PVRId

Unique ID of the PVR.

PVLId

Unique ID of the PVL.

Version

Version of the metadata structure.

RegisterBitmap

Bitmap that shows which PVR attributes the SSM has registered for change notification.

TotalCartridges

The total number of cartridges currently managed by the PVR.

CartsAlarmThreshold

The maximum number of cartridges the PVR can hold before beginning to issue "PVR almost full" alert messages.

CartsCapacity

The maximum number of cartridges the PVR can hold.

SameJobOnController

Weighted value used to select the preferred drive.

OtherJobOnController

Weighted value used to select the preferred drive.

DistanceToDrive

Weighted value used to select the preferred drive.

CharacteristicsBitmap

Bitmap which represents the state of configurable PVR settings.

CartFileName

Name of SFS file which stores cartridge data for this PVR.

DeviceSpecificA

Available for PVR specific information.

DeviceSpecificB

Available for PVR specific information.

Clients

The following clients access the data definition:

Storage System Manager.

7.3.6. Manufacturing Lot Number - lot_number_t

Description

This structure specifies the cartridge manufacturing lot number.

Format

```
typedef struct {
    idl_char LotNumber[LOT_NUMBER_SIZE];
} lot_number_t;
```

Clients

The following clients access the data definition:

Physical Volume Library, Storage System Manager

7.3.7. Cartridge Manufacturer - manufacturer_t

Description

This structure specifies the cartridge manufacturer.

Format

```
typedef struct {
    idl_char Manufacturer[MANUFACTURER_SIZE];
} manufacturer_t;
```

Clients

The following clients access the data definition:

Physical Volume Library, Storage System Manager

7.3.8. Check-in request - checkin_req_t

Description

Contains information for a check-in request.

Format

A shelf tape request has the following format:

```
typedef struct {  
    uuid_t      PVRId;  
    cart_t      Cart;  
    unsigned64  RegisterBitmap;  
} checkin_req_t;
```

PVRId

Unique id of the PVR.

Cart

Cartridge label.

RegisterBitmap

Bitmap that shows which PVR attributes the SSM has registered for change notification.

Clients

The following clients access the data definition:

Physical Volume Library.

7.3.9. Other APIs

7.3.9.1. pvl_MountCompleted

Purpose

Allows PVR to notify the PVL when a cartridge mount has completed.

Syntax

```
signed32 pvl_MountCompleted(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,     /* IN */  
    cart_t           *Cart,       /* IN */  
    side_t           *Side,       /* IN */  
    drive_t          *Drive,      /* IN */  
    job_id_t         JobId,       /* IN */  
    pvl_status       TheStatus); /* IN */
```

Description

This function should be provided by any client that calls **pvr_Mount**. The function will be called by the PVR once the cartridge is mounted. The function may be called as soon as the **pvr_Mount** function is called (possibly before the **pvr_Mount** function returns). The function will be called as an RPC by the PVR.

Parameters

| | |
|------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connect handle. |
| <i>Cart</i> | Cartridge that was mounted. |
| <i>Side</i> | Side of cartridge that was mounted. |
| <i>Drive</i> | Drive in which the cartridge was mounted. |
| <i>JobId</i> | Job associated with the mount. |
| <i>TheStatus</i> | Indicates the status of the mount. One of: SUCCESS The cartridge was mounted and is ready for read/write operations. ERROR The cartridge could not be mounted. The <i>Drive</i> argument is undefined when ERROR is returned. CANCEL The mount was canceled. The <i>Drive</i> argument is undefined when CANCEL is returned. |

Return values

Upon successful completion, the function will return a zero (0). If an error occurs, the negated error code is returned.

Error conditions

HPSS_ENOTYET The server has not completed initializing.

| | |
|-------------|---|
| HPSS_EAUTH | The client is not authorized for this API. |
| HPSS_ESRCH | The drive/cartridge/activity/job was not found. |
| HPSS_EINVAL | Invalid argument(drive was NULL). |
| HPSS_EBUSY | The drive's operational state is no longer enabled. |

See also

pvr_Mount, pvr_MountComplete.

Clients

The PVL will provide this function, the Physical Volume Repository is a client.

Notes

If an error other than HPSS_ESRCH is returned to the PVR. The PVL will elevate the cartridge from the current drive and the PVR will attempt to mount the cartridge in another drive which was provided in the mount drive list.

8. System Manager Functions

This chapter specifies the System Manager programming interface. Specifically, the following information is provided:

Application Programming Interfaces (APIs)

Data Definitions

8.1. API Functions

The System Manager supplies an interface for its clients, each of which is an SSM Data Server, to perform management tasks for the HPSS System. The Data Server establishes contact with the System Manager with **ssm_CheckIn** and disconnects with **ssm_CheckOut**. It manipulates configuration files with **ssm_ConfigAdd**, **ssm_ConfigDelete**, **ssm_ConfigGetDefault**, **ssm_ConfigRead**, and **ssm_ConfigUpdate**. It starts, stops, reinitializes, repairs, and connects to servers with **ssm_Adm**. It monitors and manipulates managed objects managed by other HPSS servers with **ssm_AttrGet**, **ssm_AttrReg**, and **ssm_AttrSet**. It manages storage media with **ssm_CartImport**, **ssm_CartExport**, **ssm_CartMove**, **ssm_ResourceCreate**, **ssm_ResourceDelete**, **ssm_ResourceRepack**, and **ssm_ResourceReclaim**. It performs other system administration tasks with **ssm_DriveDismount**, **ssm_JobCancel**, **ssm_Delog**, **ssm_AcctChange**, and **ssm_AcctRun**. The System Manager also performs fileset and junction management functions using **ssm_FilesetCreate**, **ssm_FilesetDelete**, **ssm_JunctionCreate** and **ssm_JunctionDelete**.

This section describes all API interfaces which are provided for use by another HPSS subsystem or by a client external to HPSS. The API interface specification includes the following information:

Name

Syntax

Description

Parameters

Return Values

Error Conditions

Related Information

Clients

Notes

8.1.1. `ssm_AcctChange`

Purpose

Change the account ID on a file.

Syntax

```
#include "ssm_types.h"

signed32 ssm_AcctChange(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    unsigned32       ClientID,    /* IN */
    SrvInfoUnion_t  *ObjectID,   /* IN */
    acct_rec_t       AcctID,     /* IN */
    unsigned32       *RPCStatus); /* OUT */
```

Description:

Changes the account ID on the specified bitfile.

Parameters

| | |
|------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientID</i> | Index into Client List. |
| <i>ObjectID</i> | ID of the object (file) for which to change the account. |
| <i>AcctID</i> | New account ID to assign to the object. |
| <i>RPCStatus</i> | DCE return code. |

Return values

Upon successful completion, the operation returns 0. Otherwise, the appropriate error code is returned.

Error conditions

| | |
|------------------------|--|
| HPSS_ENOTREADY | System Manager is not yet initialized. |
| SSM_EACCESS | Client does not have permission for operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_OBJID | Invalid object ID for bitfile. |
| SSM_EINVALID_IN | Invalid input. |

See also

`ssm_AcctRun`.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.2. `ssm_AcctRun`

Purpose

Run HPSS accounting

Syntax

```
signed32 ssm_AcctRun(  
    handle_t                Bh,                /* IN */  
    hpss_connect_handle_t  *Ch,                /* IN */  
    unsigned32             ClientID,          /* IN */  
    unsigned32             *RPCStatus);      /* OUT */
```

Description:

Start the accounting program. Refuse to run it if no accounting policy exists.

Parameters

| | |
|------------------|---------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientID</i> | Index into SSM_SM_client. |
| <i>RPCStatus</i> | DCE return code. |

Return values

Upon successful completion, the operation returns 0. Otherwise, the appropriate error code is returned.

Error conditions

| | |
|------------------------|--|
| HPSS_ENOTREADY | System Manager is not yet initialized. |
| SSM_EACCESS | Client does not have permission for operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_ECANT_MALLOC | Can't malloc. |
| SSM_ESM_INTERNAL_ERROR | Fork or exec failed. |
| SSM_ENO_ACCT_POLICY | No accounting policy has yet been defined. |

See also

`ssm_AcctChange`.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_client_if.acf` file private to the calling application.

8.1.3. `ssm_Adm`**Purpose**

Perform administrative requests on one or more servers.

Syntax

```
#include "ssm_types.h"
signed32 ssm_Adm(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    unsigned32         OperationID, /* IN */
    unsigned32         ServerID,    /* IN */
    unsigned32         ClientID,    /* IN */
    unsigned32         *RPCStatus); /* OUT */
```

Description:

Processes requests from clients to perform administrative functions, including starting, reinitializing, repairing, connecting to, and stopping one or all subsystems. If *ServerID* is `SSM_MAX_SERVER`, the operation will be attempted for all servers.

Supported operations:

| | |
|-----------------------------------|--|
| <code>SSM_ADM_HPSS_SHUT</code> | Shutdown HPSS. Perform an orderly shutdown of all the servers except SSM and the Startup Daemon. |
| <code>SSM_ADM_SRV_HALT</code> | Halt one or more servers. Request that the server change the <i>AdministrativeState</i> of his server managed object to <code>ST_HALT</code> and exit. In addition, ask the startup demon to send him a kill signal. |
| <code>SSM_SRV_CONNECT</code> | Signal the thread monitoring this server to awaken and try to connect to him. |
| <code>SSM_ADM_SRV_REINIT</code> | Reinitialize one or all servers. Request that the server change the <i>AdministrativeState</i> of his server managed object to <code>ST_REINIT</code> and reread his configuration files. |
| <code>SSM_ADM_SRV_REPAIRED</code> | Notify a server of repair. Request that the server change the <i>AdministrativeState</i> of his server managed object to <code>ST_REPAIRED</code> and clear his error statuses. |
| <code>SSM_ADM_SRV_SHUT</code> | Shutdown one or all servers. Request that the server change the <i>AdministrativeState</i> of his server managed object to <code>ST_SHUTDOWN</code> and proceed with an orderly shutdown. |
| <code>SSM_ADM_SRV_START</code> | Start one or all servers. |

Parameters

| | |
|--------------------|-------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>OperationID</i> | Operation code. |

| | |
|------------------|--|
| <i>ServerID</i> | Server ID; index of server in the Server List supplied by the System Manager to its clients. |
| <i>ClientID</i> | Client ID; index of client in the System Manager's list of clients. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

For the operations of starting, shutting down, or halting all servers, the function returns a positive number indicating the number of servers for which the operation successfully completed. Upon successful completion for other operations, the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTSUPPORTED | Operation not supported by server. |
| HPSS_EALREADY_RUNNING | A request was made to start a server that was already running. |
| HPSS_ENOTREADY | The System Manager is not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_EINVALID_OP | Invalid operation code. |
| SSM_ESHUT_FAIL | Shutdown failed. |
| SSM_ECANT_MALLOC | Can't malloc. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable timed out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_ESTART_NO_EXEC | The EXECUTE_SERVER_FLAG bit is not set in the <i>Flags</i> field of the server in the HPSS Server Configuration file. |
| SSM_EBIND_FAIL | The System Manager was unable to bind to either the server or its startup demon. |
| SSM_EINVALID_MO | Invalid managed object (internal System Manager failure). |
| SSM_EINVALID_SRVID | Invalid server ID. |
| SSM_EINVALID_SRVTYPE | Invalid server type (internal System Manager failure). |
| SSM_ENO_STARTUP | The startup demon for the server's system could not be found in the HPSS Server Configuration file. |

| | |
|--------------------|-------------------|
| SSM_ERPC_ERROR | RPC error. |
| SSM_ESRV_NOT_FOUND | Server not found. |
| SSM_ESTART_FAIL | Startup failed. |

See also

None.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

Currently, the SSM Data Server does not use the "ALL" capability of this operation. It requests each operation only on one server at a time.

SSM cannot shut down the Startup Daemon.

8.1.4. `ssm_AttrGet`

Purpose

Get managed object attributes.

Syntax

```
#include "ssm_types.h"

signed32 ssm_AttrGet(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    unsigned32         ServerID,    /* IN */
    unsigned32         ClientID,    /* IN */
    unsigned32         MOCClass,    /* IN */
    SrvInfoUnion_t    *ObjectID,   /* IN */
    SrvInfoUnion_t    *OutData,    /* OUT */
    unsigned32        *RPCStatus); /* OUT */
```

Description:

The function `ssm_AttrGet` retrieves the attributes of the requested managed object.

Parameters

| | |
|------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ServerID</i> | Server ID; index of server in the Server List supplied by the System Manager to its clients. |
| <i>ClientID</i> | Client ID; index of client in the System Manager's list of clients. |
| <i>MOCClass</i> | Type of managed object requested. |
| <i>ObjectID</i> | Object ID. |
| <i>OutData</i> | Returned managed object. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion `ssm_AttrGet` returns the value returned from the requested server API. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_EINVALID_OPMO | Invalid operation for the managed object. |
| SSM_ECLIENT_NOT_REG | Client is not registered. |

| | |
|-------------------------|--|
| SSM_ECANT_FIND_PVR | The System Manager cannot determine which PVR controls the requested cartridge. |
| SSM_ECANT_FIND_SS | The System Manager cannot determine which Storage Server controls the requested volume. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_EINVALID_SRVTYPE | Invalid server type found in System Manager's server list; this is an internal system manager problem. |
| SSM_EINVALID_MO | Invalid managed object type. |
| SSM_EINVALID_OBJID | Invalid object ID. |
| SSM_EINVALID_SRVID | Invalid server ID. |
| SSM_EINVALID_MO_SRVTYPE | Invalid managed object type for this server type. |
| SSM_ECANT_MALLOC | Can't malloc. |
| SSM_EBIND_FAIL | Can't bind to server. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_ERPC_ERROR | RPC error. |

Return code from subsystem's get attribute API.

See also

ssm_AttrSet, ssm_AttrReg.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.5. ssm_AttrReg

Purpose

Register an SSM client to receive notifications of data changes in managed objects.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_AttrReg(  
    handle_t                Bh,                /* IN */  
    hpss_connect_handle_t  *Ch,                /* IN */  
    unsigned32             ServerID,          /* IN */  
    unsigned32             ClientID,          /* IN */  
    unsigned32             MOCClass,          /* IN */  
    SrvInfoUnion_t        *ObjectID,          /* IN */  
    u_signed64             RegisterBitmap,    /* IN */  
    SrvInfoUnion_t        *OutData,          /* OUT */  
    unsigned32             *RPCStatus);      /* OUT */
```

Description

The function **ssm_AttrReg** registers with the specified server to receive notifications of data changes in the specified managed object. A current copy of the managed object is returned in *OutData*.

Parameters

| | |
|-----------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ServerID</i> | Server ID; index of server in the Server List supplied by the System Manager to its clients. |
| <i>ClientID</i> | Client ID; index of client in the System Manager's list of clients. |
| <i>MOCClass</i> | Type of managed object. |
| <i>ObjectID</i> | Object ID. |
| <i>RegisterBitmap</i> | Fields for which to register. |
| <i>OutData</i> | The complete new managed object. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion **ssm_AttrRet** returns the value returned from the set attribute API for the requested managed object. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |

| | |
|----------------------|--|
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_ECANT_FIND_PVR | The System Manager can't determine the PVR to which the requested cartridge belongs. |
| SSM_ECANT_FIND_SS | The System Manager can't determine the Storage Server to which the requested volume belongs. |
| SSM_EBIND_FAIL | System manager cannot bind to server. |
| SSM_ECANT_MALLOC | Can't malloc space for input object. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_EINVALID_IN | Invalid input; this is an internal System Manager problem. |
| SSM_EINVALID_MO | Invalid managed object code. |
| SSM_EINVALID_OBJID | Invalid object ID. |
| SSM_EINVALID_SRVID | Invalid server ID. |
| SSM_EINVALID_SRVTYPE | Invalid server type (internal System Manager failure). |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_ENULL_PTR | Null pointer; this is an internal system manager problem. |
| SSM_ERPC_ERROR | RPC error. |
| SSM_ESRV_NOT_FOUND | Server not found. |

Return code from subsystem's set attribute API.

See also

ssm_AttrGet, ssm_AttrSet.

Clients

SSM Data Server.

Notes

The returned managed object will contain a *RegisterBitmap* for all notifications for which the managed object is registered, which might include registrations from clients other than the one making this request.

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_client_if.acf` file private to the calling application.

8.1.6. `ssm_AttrSet`

Purpose

Set managed object attributes.

Syntax

```
#include "ssm_types.h"

signed32 ssm_AttrSet(
    handle_t                Bh,           /* IN */
    hpss_connect_handle_t  *Ch,         /* IN */
    unsigned32             ServerID,    /* IN */
    unsigned32             ClientID,    /* IN */
    SrvInfoUnion_t        *ObjectID,    /* IN */
    u_signed64             InBitmap,    /* IN */
    u_signed64             *OutBitmap,   /* OUT */
    SrvInfoUnion_t        *InData,     /* IN */
    SrvInfoUnion_t        *OutData,    /* OUT */
    unsigned32             *RPCStatus); /* OUT */
```

Description

The function `ssm_AttrSet` sets the attributes of the specified managed object.

Parameters

| | |
|------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ServerID</i> | Server ID; index of server in the Server List supplied by the System Manager to its clients. |
| <i>ClientID</i> | Client ID; index of client in the System Manager's list of clients. |
| <i>ObjectID</i> | Object ID. |
| <i>InBitmap</i> | Fields to set. |
| <i>OutBitmap</i> | Fields actually set. |
| <i>InData</i> | Managed object holding values to set. |
| <i>OutData</i> | Managed object holding values actually set. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion `ssm_AttrSet` returns the value returned from the requested server API. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |

| | |
|----------------------|---|
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_ECANT_FIND_PVR | The System Manager cannot determine which PVR controls the requested cartridge. |
| SSM_ECANT_FIND_SS | The System Manager cannot determine which Storage Server controls the requested volume. |
| SSM_EBIND_FAIL | System manager cannot bind to server. |
| SSM_ECANT_MALLOC | Can't malloc. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_EINVALID_MO | Invalid managed object. |
| SSM_EINVALID_OBJID | Invalid object ID. |
| SSM_EINVALID_SRVID | Invalid server ID. |
| SSM_EINVALID_SRVTYPE | Invalid server type. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_ENULL_PTR | Invalid null pointer (internal System Manager failure). |
| SSM_ERPC_ERROR | RPC error. |
| SSM_ESRV_NOT_FOUND | Server not found. |

Return code from subsystem's set attributes API.

See also

ssm_AttrGet, ssm_AttrReg.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.7. `ssm_CartExport`

Purpose

Export cartridges from the PVL.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_CartExport(  
    handle_t           Bh,           /* IN */  
    hpss_connect_handle_t *Ch,      /* IN */  
    unsigned32 ClientID,           /* IN */  
    unsigned32 PVLID,             /* IN */  
    PvlCartInfo_t *CartData,       /* IN */  
    unsigned32 *NumProcessed,      /* OUT */  
    unsigned32 *RPCStatus);        /* OUT */
```

Description:

The `ssm_CartExport` function exports a list of cartridges from the PVL.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | DCE binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientID</i> | Index of client into Client List. |
| <i>PVLID</i> | PVL ID. |
| <i>CartData</i> | Identifying information for list of cartridges to be exported. It includes a list of cartridges and the total number of cartridges. |
| <i>NumProcessed</i> | Number of cartridges successfully exported. |
| <i>RPCStatus</i> | RPC error code, set by DCE. |

Return values

Upon successful completion the function returns 0 and *NumProcessed* will be equal to *CartData.NumCarts*. If an error occurs, the negated error code is returned and *NumProcessed* is set to the array index of the cartridge on which the error occurred.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |

The return code from `pvl_Export`

See also

ssm_CartImport, ssm_CartMove, ssm_ResourceCreate, ssm_ResourceDelete.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.8. `ssm_CartImport`

Purpose

Import cartridges into the PVL.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_CartImport(  
    handle_t                Bh,                /* IN */  
    hpss_connect_handle_t  *Ch,                /* IN */  
    unsigned32              ClientID,          /* IN */  
    unsigned32              PVLID,            /* IN */  
    PvlCartInfo_t          *CartData,          /* IN */  
    PvlImport_t            *ImportData,        /* IN */  
    unsigned32              *NumProcessed,     /* OUT */  
    unsigned32              *RPCStatus);       /* OUT */
```

Description:

The `ssm_CartImport` function imports the list of cartridges specified into the PVL.

Parameters

| | | | | | | | | | | | | | | | |
|---------------------|---|-------|--------------------------------|-----------|-------------------------------------|-----------|-------------|------------|--------------------------------|-------|--------------------------------|------|---------------|-----|------|
| <i>Bh</i> | DCE binding handle. | | | | | | | | | | | | | | |
| <i>Ch</i> | HPSS connection handle. | | | | | | | | | | | | | | |
| <i>ClientID</i> | Client ID. | | | | | | | | | | | | | | |
| <i>PVLID</i> | PVL ID. | | | | | | | | | | | | | | |
| <i>CartData</i> | Identifying information for the list of cartridges to be imported, including the names and the total number in the list. | | | | | | | | | | | | | | |
| ImportData | Data required by the PVL for importing cartridges. It includes: <table><tr><td>PVRID</td><td>Index of PVR into Server List.</td></tr><tr><td>MaxDrives</td><td>Maximum drives to devote to import.</td></tr><tr><td>MediaType</td><td>Media type.</td></tr><tr><td>ImportType</td><td>Import type (Scratch or HPSS).</td></tr><tr><td>Sides</td><td>Number of sides or partitions.</td></tr><tr><td>Manu</td><td>Manufacturer.</td></tr><tr><td>Lot</td><td>Lot.</td></tr></table> | PVRID | Index of PVR into Server List. | MaxDrives | Maximum drives to devote to import. | MediaType | Media type. | ImportType | Import type (Scratch or HPSS). | Sides | Number of sides or partitions. | Manu | Manufacturer. | Lot | Lot. |
| PVRID | Index of PVR into Server List. | | | | | | | | | | | | | | |
| MaxDrives | Maximum drives to devote to import. | | | | | | | | | | | | | | |
| MediaType | Media type. | | | | | | | | | | | | | | |
| ImportType | Import type (Scratch or HPSS). | | | | | | | | | | | | | | |
| Sides | Number of sides or partitions. | | | | | | | | | | | | | | |
| Manu | Manufacturer. | | | | | | | | | | | | | | |
| Lot | Lot. | | | | | | | | | | | | | | |
| <i>NumProcessed</i> | Number of cartridges processed successfully. | | | | | | | | | | | | | | |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. | | | | | | | | | | | | | | |

Return values

Upon successful completion the function returns 0 and *NumProcessed* will be equal to *CartData.NumCarts*. If an error occurs, the negated error code is returned and *NumProcessed* is set to the number (the array index) of the cartridge on which the error occurred.

ssm_CartImport creates up to *MaxDrives* threads each of which processes a portion of the cartridge list concurrently. Whether *MaxDrives* cartridges will actually be mounted and processed concurrently depends upon the number of drives available to the system and the load from other jobs.

If a specified cartridge has already been imported, this routine returns success for it.

Error conditions

| | |
|------------------------|--|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation.. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_ECANT_MALLOC | Can't malloc. |
| SSM_ESM_INTERNAL_ERROR | Can't create threads or join with them. |

The return code from **pvl_Import**.

See also

ssm_CartExport, **ssm_CartMove**, **ssm_ResourceCreate**, **ssm_ResourceDelete**.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.9. `ssm_CartMove`

Purpose

Move a cartridge from one PVR to another.

Syntax

```
#include "ssm_types.h"

signed32 ssm_CartMove(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    unsigned32       ClientID,    /* IN */
    unsigned32       PVLID,       /* IN */
    uuid_t           *PVRID,      /* IN */
    cart_t           *Cart,        /* IN */
    unsigned32       *RPCStatus); /* OUT */
```

Description:

The `ssm_CartMove` function moves a cartridge from one PVR to another. Only the target PVR is specified, as the PVL should be able to figure out the PVR in which the cartridge currently resides.

Parameters

| | |
|------------------|-----------------------------------|
| <i>Bh</i> | DCE binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientID</i> | Client ID. |
| <i>PVLID</i> | PVL ID. |
| <i>PVRID</i> | Destination PVR. |
| <i>Cart</i> | Cartridge ID. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_ESRV_NOT_FOUND | Server not found. |

The return code from `pvl_Move`.

See also

`ssm_CartExport`, `ssm_CartImport`, `ssm_ResourceCreate`, `ssm_ResourceDelete`.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_client_if.acf` file private to the calling application.

8.1.10. `ssm_CheckIn`

Purpose

Accept check-ins from Data Server clients.

Syntax

```
#include "ssm_types.h"

signed32 ssm_CheckIn(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t Ch,      /* IN */
    char              ClientName, /* IN */
    unsigned32        ListFlags,   /* IN */
    unsigned32        InClientID,  /* IN */
    unsigned32        OutClientID, /* OUT */
    unsigned32        RPCStatus); /* OUT */
```

Description:

The `ssm_CheckIn` function accepts check-in requests from clients. On a client's first call to `ssm_CheckIn`, the client supplies `SSM_NEW_CLIENT` as the *InClientID*. The System Manager adds the client to its list of clients and returns the *OutClientID*.

On subsequent calls by a client to `ssm_CheckIn`, the client supplies its valid client ID as *InClientID*.

On every check-in, the System Manager asynchronously sends the client a current copy of every shared list (the Server List, Driver List, Class of Service List, Storage Class List, Hierarchy List, Migration Policy List, and Purge Policy List).

Parameters

| | |
|--------------------|---|
| <i>Bh</i> | RPC handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientName</i> | Client's CDS name. |
| <i>ListFlags</i> | A bitmask representing which lists the client wants. |
| <i>InClientID</i> | Client's index into the System Manager's list of known clients; supplied by client on all calls to <code>ssm_CheckIn</code> except first; required from client in all subsequent communication with the System Manager. |
| <i>OutClientID</i> | Supplied by server to client on first call to <code>ssm_CheckIn</code> only. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|-----------------------------|---|
| <code>HPSS_ENOTREADY</code> | The System Manager has not yet initialized. |
| <code>SSM_EACCESS</code> | The client does not have permission for this operation. |

| | |
|------------------------|---------------------------------|
| SSM_ECLIENT_NOT_REG | Client never checked in before. |
| SSM_EINVALID_CLIENT_ID | Invalid client ID. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_ENO_CLIENT_SLOTS | No more client slots available. |

See also

`ssm_CheckOut`.

Clients

SSM Data Server.

Notes

The *ListFlags* parameter is intended for future use and is ignored by the System Manager.

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_client_if.acf` file private to the calling application.

8.1.11. `ssm_CheckOut`

Purpose

Accept checkouts from SSM Data Server clients.

Syntax

```
#include "ssm_types.h"

signed32 ssm_CheckOut(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    unsigned32        ClientID,   /* IN */
    unsigned32        *RPCStatus); /* OUT */
```

Description

The `ssm_CheckOut` function accepts check-out requests from clients. It removes the client from the System Manager's list of clients and from the table of registered attributes, and frees the binding handle for the client.

Clients who cannot be reached by the System Manager within a set time limit are checked out automatically.

Parameters

| | |
|------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientID</i> | Client ID; index of client in the System Manager's list of clients. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | | |
|------------------------|---|--------|
| HPSS_ENOTREADY | The System Manager has not yet initialized. | |
| SSM_EACCESS | The client does not have permission for this operation. | |
| SSM_ECLIENT_NOT_REG | Client never checked in. | |
| SSM_EINVALID_CLIENT_ID | Invalid client ID. | |
| SSM_ECONDITION_FAIL | Condition variable failure. | |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. | |
| SSM_EMUTEX_FAIL | Mutex failure. | |
| SSM_ECANT_MALLOC | Can't malloc. | |
| SSM_ESM_INTERNAL_ERROR | Manager internal error. | System |

See also

ssm_CheckIn.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.12. `ssm_ConfigAdd`

Purpose

Add new entries to configuration files.

Syntax

```
signed32 ssm_ConfigAdd(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    unsigned32        FileOwner, /* IN */  
    unsigned32        ClientID,   /* IN */  
    SrvInfoUnion_t    *InData,   /* IN */  
    unsigned32        *RPCStatus); /* OUT */
```

Description:

The `ssm_ConfigAdd` function adds a new entry to a configuration file.

Parameters

| | |
|------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>FileOwner</i> | ID of the server which references the configuration file; index of server in the Server List supplied by the System Manager to its clients. |
| <i>ClientID</i> | Client ID; index of client in the System Manager's list of clients. |
| <i>InData</i> | New configuration file entry to create. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|--|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_ESRV_MUST_BE_DOWN | Server must be down to change its configuration. |
| SSM_EOPEN_FAIL | Failure opening SFS file. |
| SSM_ENO_ACCT_POLICY | Specified accounting policy style is invalid. |
| SSM_ESCFIRST_SSM | A server of type SSM must be added to the generic configuration before any other type. |

| | |
|----------------------|--|
| SSM_EINVALID_OPCF | Invalid operation for this configuration file; trying to add either a device or drive file without adding the other. |
| SSM_ETRANABORT | Transaction aborted. |
| SSM_EALREADY_IN_CF | Entry was already in configuration file. |
| SSM_EMMFAIL | Metadata manager failure. |
| SSM_ECDS_CREATE_FAIL | When a server was being added to the generic configuration file, the creation of the CDS directory, the creation of its Security object, and/or the setting of the ACLs on one or both failed. However, if the System Manager got this far, the requested entry was added to the configuration file. |
| SSM_ENULL_PTR | Invalid input; a null pointer was supplied where a value was required. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_EINVALID_CF | Invalid configuration file type. |
| SSM_ECF_LIST_BAD | Although the Add operation completed successfully, the System Manager's internal list of configuration files is corrupted. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_ENO_SUCH_CF | No such configuration file. |

See also

ssm_ConfigDelete, ssm_ConfigGetDefault, ssm_ConfigRead, ssm_ConfigUpdate.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.13. `ssm_ConfigDelete`

Purpose

Delete an entry from a configuration file.

Syntax

```
#include "ssm_types.h"
signed32 ssm_ConfigDelete(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    unsigned32       FileOwner,  /* IN */
    unsigned32       ClientID,   /* IN */
    unsigned32       CFClass,    /* IN */
    SrvInfoUnion_t  *ObjectID,   /* IN */
    unsigned32       *RPCStatus); /* OUT */
```

Description

The `ssm_ConfigDelete` function deletes an entry from a configuration file.

Parameters

| | |
|------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>FileOwner</i> | ID of the server which references the configuration file; index of server in the Server List supplied by the System Manager to its clients. |
| <i>ClientID</i> | Client ID; index of client in the System Manager's list of clients. |
| <i>CFClass</i> | Type of configuration file to update. |
| <i>ObjectID</i> | Index key of entry to delete. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_ESRV_MUST_BE_DOWN | Server must be down during this operation. |
| SSM_ESM_INTERNAL_ERROR | Internal System Manager error. |

| | |
|-------------------------|---|
| SSM_ENO_SUCH_CF | No such configuration file. |
| SSM_EOPEN_FAIL | Cannot open file. |
| SSM_EDEV_PARTNER_EXISTS | Attempt to delete a drive when the corresponding device still exists, or a device when the corresponding drive still exists. |
| SSM_EINVALID_CF | Invalid configuration file type. |
| SSM_ETRANABORT | Transaction aborted. |
| SSM_ENOT_IN_CF | Entry is not in configuration file. |
| SSM_EMMFAIL | Metadata manager failure. |
| SSM_ECANT_MALLOC | Can't malloc. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_EINVALID_OBJID | Invalid object ID. |
| SSM_ESRV_NOT_FOUND | Server not found. |
| SSM_ECF_LIST_BAD | Although the Delete operation completed successfully, the System Manager's internal list of configuration files is corrupted. |
| SSM_ENO_SUCH_DEVICE | device not found in configuration file. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out |
| SSM_EINVALID_OPCF | Invalid operation for this configuration type; trying to delete a device entry without deleting the corresponding drive entry, or vice versa. |
| SSM_EMUTEX_FAIL | Mutex failure. |

See also

ssm_ConfigAdd, ssm_ConfigGetDefault, ssm_ConfigRead, ssm_ConfigUpdate.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.14. ssm_ConfigGetDefault**Purpose**

Obtain a default entry for a configuration file.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_ConfigGetDefault(  
    handle_t          Bh,          /* IN */  
    hpss_connect_handle_t *Ch,    /* IN */  
    unsigned32       ClientID,    /* IN */  
    unsigned32       *ServerType, /* IN */  
    unsigned32       *ServerSubType, /* IN */  
    unsigned32       ServerID,    /* IN */  
    unsigned32       CFClass,     /* IN */  
    SrvInfoUnion_t  *Entry,      /* OUT */  
    unsigned32      *RPCStatus); /* OUT */
```

Description:

The **ssm_ConfigGetDefault** function gets a default entry for an HPSS configuration file.

For Server Configuration File entries, a valid *ServerType*, and *ServerSubType* if applicable, must be specified. Currently, *SubType* applies only to Storage Servers and PVRs.

For specific server configuration files, such as the BFS Configuration File, a valid *ServerID* must be supplied. The returned entry will include the descriptive name and uuid of the specified server.

Parameters

| | |
|----------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientID</i> | Index of client into Client List. |
| <i>ServerType</i> | Type of server; ignored if not needed. |
| <i>ServerSubType</i> | SubType of server; ignored if not needed. |
| <i>ServerID</i> | Server ID; index of server in the Server List supplied by the System Manager to its clients; SSM_MAX_SERVERS if not needed. |
| <i>CFClass</i> | Type of configuration file. |
| <i>Entry</i> | The returned default entry. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. In the event the specific server configuration file already has an entry for that server, **ssm_ConfigGetDefault** returns the existing entry instead of creating a new default entry, and returns SSM_ESRV_ALREADY_IN_CF. If an error occurs, the negated error code is returned.

Error conditions

| | |
|-------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CF | Invalid configuration file type. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_EINVALID_OPCF | Invalid operation for this configuration type. |
| SSM_EOPEN_FAIL | Could not open configuration file. |
| SSM_EALREADY_IN_CF | Entry was already in configuration file. |
| SSM_EMMFAIL | Metadata manager failure. |
| SSM_ESCFIRST_SSM | SSM must be the first entry added to the file. |
| SSM_ESCFIRST_BFS_DEPEND | The MPS, Name Server, and Storage Server entries must be added to the generic server configuration file before the BFS Specific Configuration entry may be created. |
| SSM_ESCFIRST_MPS_DEPEND | The BFS and Storage Server entries must be added to the generic server config file before the MPS Specific Configuration entry may be created. |
| SSM_ESCFIRST_NS_DEPEND | The BFS entry must be added to the generic server config file before the Name Server Specific Configuration entry may be created. |
| SSM_ESCFIRST_PVR_DEPEND | The PVL entry must be added to the generic server configuration file before the PVR Specific Configuration entry may be created. |
| SSM_ESCFIRST_SS_DEPEND | The PVL entry must be added to the generic server configuration file before the Storage Server Specific Configuration entry may be created. |
| SSM_ETRANABORT | Transaction aborted. |
| SSM_ENO_SUCH_CF | No such configuration file. |

See also

ssm_ConfigAdd, ssm_ConfigDelete, ssm_ConfigRead, ssm_ConfigUpdate.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.15. **ssm_ConfigRead****Purpose**

Read configuration file entry.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_ConfigRead(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    unsigned32        FileOwner,    /* IN */
    unsigned32        ClientID,     /* IN */
    unsigned32        CFClass,      /* IN */
    SrvInfoUnion_t   *ObjectID,     /* IN */
    u_signed64        *BfOffset,     /* IN */
    signed32          *BfStorageType, /* IN */
    SrvInfoUnion_t   *OutData,     /* OUT */
    unsigned32        *RPCStatus);  /* OUT */
```

Description:

The **ssm_ConfigRead** function reads an entry from a configuration file.

Parameters

| | |
|----------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>FileOwner</i> | ID of the server which references the configuration file; index of server in the Server List supplied by the System Manager to its clients. |
| <i>ClientID</i> | Client ID; index of client in the System Manager's list of clients. |
| <i>CFClass</i> | Type of configuration file. |
| <i>ObjectID</i> | Index key of entry to read. |
| <i>BfOffset</i> | Offset; used for bitfile segment only. |
| <i>BfStorageType</i> | Storage type; used for bitfile segment only. |
| <i>OutData</i> | Returned configuration file entry. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|----------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |

| | |
|------------------------|--|
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EOPEN_FAIL | Can't open file. |
| SSM_EINVALID_OPCF | Invalid operation for this configuration type. |
| SSM_EINVALID_CF | Invalid configuration file type. |
| SSM_EMMFAIL | Metadata manager failure. |
| SSM_ETRANABORT | Transaction aborted. |
| SSM_ENOT_IN_CF | Entry is not in configuration file. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_EINVALID_OBJID | Invalid object ID. |
| SSM_ECANT_MALLOC | Can't malloc. |
| SSM_ENO_SUCH_CF | No such configuration file. |

See also

ssm_ConfigAdd, ssm_ConfigDelete, ssm_ConfigGetDefault, ssm_ConfigUpdate.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.16. ssm_ConfigUpdate**Purpose**

Update configuration file entries.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_ConfigUpdate(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    unsigned32         FileOwner,   /* IN */
    unsigned32         ClientID,    /* IN */
    SrvInfoUnion_t    *InData,     /* IN */
    unsigned32         AttrList,    /* IN */
    unsigned32         *RPCStatus); /* OUT */
```

Description:

The **ssm_ConfigUpdate** function updates a configuration file entry.

Parameters

| | |
|------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>FileOwner</i> | ID of the server which references the configuration file; index of server in the Server List supplied by the System Manager to its clients. |
| <i>ClientID</i> | Client ID; index of client in the System Manager's list of clients. |
| <i>InData</i> | Entry to update. |
| <i>AttrList</i> | Bitmask representing fields to update. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_ESRV_MUST_BE_DOWN | Server must be down to update configuration. |
| SSM_ESM_INTERNAL_ERROR | System Manager internal error. |

| | |
|----------------------|---|
| SSM_EOPEN_FAIL | Can't open file. |
| SSM_EINVALID_OPCF | Invalid operation for this configuration file. |
| SSM_ETRANABORT | Transaction aborted. |
| SSM_EINVALID_CF | Invalid configuration file type. |
| SSM_ENOT_IN_CF | Entry is not in configuration file. |
| SSM_EMMFAIL | Metadata manager failure. |
| SSM_ECDS_CREATE_FAIL | When a server was being updated in the generic configuration file, and its CDS directory has been changed, the creation of the new CDS directory, the creation of its Security object, and/or the setting of the ACLs on one or both failed. However, if the System Manager got this far, the requested entry was updated properly in the configuration file. |
| SSM_ENO_SUCH_CF | No such configuration file. |
| SSM_ECF_LIST_BAD | Although the Update operation completed successfully, the System Manager's internal list of configuration files is corrupted. |
| SSM_ENULL_PTR | Invalid null pointer. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_ESRV_NOT_FOUND | Server not found. |
| SSM_ECANT_MALLOC | Can't malloc. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_EINVALID_OBJID | Invalid object ID. |
| SSM_EMUTEX_FAIL | Mutex failure. |

See also

ssm_ConfigAdd, ssm_ConfigDelete, ssm_ConfigGetDefault, ssm_ConfigRead.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.17. **ssm_Dellog****Purpose**

Allow access to delog command from Sammi.

Syntax

```
#include "ssm_types.h"

signed32 ssm_Dellog(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    unsigned32       ClientID,    /* IN */
    char             *InputFile,   /* IN */
    char             *OutputFile,  /* IN */
    idl_long_int     StartTime,    /* IN */
    idl_long_int     EndTime,      /* IN */
    idl_long_int     RecordTypes, /* IN */
    char             DescNames[SSM_MAX_DELOG_SERVERS]
                        [HPSS_MAX_DESC_NAME], /* IN */
    char             Users[SSM_MAX_DELOG_USERS]
                        [HPSS_MAX_DESC_NAME], /* IN */
    unsigned32       *RPCStatus); /* OUT */
```

Description:

The **ssm_Dellog** function executes the HPSS delog utility. Output from the delog utility is placed in *OutputFile*. The standard error from the delog program is placed in "*OutputFile*".err.

Parameters

| | |
|--------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientID</i> | Client ID; index of client in the System Manager's list of clients |
| <i>InputFile</i> | Input file (required). |
| <i>OutputFile</i> | Output file (required). |
| <i>StartTime</i> | Starting time. |
| <i>EndTime</i> | Ending time. |
| <i>RecordTypes</i> | Record types for filtering. |
| <i>DescNames</i> | Descriptive names for filtering. |
| <i>Users</i> | User Names for filtering. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If the delog program is terminated by a signal, the function returns the positive value of the signal. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| positive number | Value of signal which terminated delog program. |
| -1 | Failure from delog program. |
| SSM_ECANT_MALLOC | Can't malloc. |
| SSM_EOPEN_FAIL | Can't open file for standard error. |
| SSM_ESM_INTERNAL_ERROR | Fork or exec failed. |

See also

None.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.18. ssm_DriveDismount**Purpose**

Dismount a drive.

Syntax

```
#include "ssm_types.h"

signed32 ssm_DriveDismount(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    unsigned32       ClientID,    /* IN */
    unsigned32       PVLID,      /* IN */
    drive_t          Drive,       /* IN */
    unsigned32       *RPCStatus); /* OUT */
```

Parameters

| | |
|------------------|---------------------------------|
| <i>Bh</i> | RPC binding handle |
| <i>Ch</i> | HPSS connection handle |
| <i>ClientID</i> | Index into Client List |
| <i>PVLID</i> | PVL ID |
| <i>Drive</i> | Drive to be dismounted |
| <i>RPCStatus</i> | RPC error code, supplied by DCE |

Description:

The **ssm_DriveDismount** function allows an operator to force a cartridge to be dismounted from a drive. It is provided for the operator to correct the situation in which HPSS does not, for some reason, dismount the drive automatically.

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_EBIND_FAIL | System manager could not bind to server. |
| SSM_ERPC_ERROR | RPC error. |

The return code from **pvl_DismountDrive**.

See also

None.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.19. ssm_FilesetCreate**Purpose**

Create a DFS/HPSS or HPSS-only fileset.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_FilesetCreate (
    handle_t           Bh,           /*IN*/
    hpss_connect_handle_t *Ch,       /*IN*/
    unsigned32        DMGID,        /*IN*/
    unsigned32        ClientID,     /*IN*/
    SrvInfoUnion_t    *InFileset,    /*IN*/
    SrvInfoUnion_t    *OutFileset,   /*OUT*/
    unsigned32        *RPCStatus);  /*OUT*/
```

Description

The **ssm_FilesetCreate** function is used to create DFS/HPSS and HPSS-only filesets.

Parameters

| | |
|-------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>DMGID</i> | Index into Server List for the DMG to be asked to create the fileset. |
| <i>ClientID</i> | Index into the Client List. |
| <i>InFileset</i> | Fileset to be created. |
| <i>OutFileset</i> | Full fileset structure returned. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_ID | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_BIND_FAIL | System Manager cannot bind to server. |
| SSM_ERPC_ERROR | RPC error. |

The return value from `dmg_admin_FilesetCreate` or `hpss_FilesetCreate`.

See also

`ssm_FilesetDelete`, `dmg_admin_FilesetCreate`, `hpss_FilesetCreate`.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_client_if.acf` file private to the calling application.

8.1.20. **ssm_FilesetDelete****Purpose**

Delete a DFS/HPSS or HPSS-only fileset.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_FilesetDelete (
    handle_t           Bh,           /*IN*/
    hpss_connect_handle_t *Ch,       /*IN*/
    unsigned32        DMGID,       /*IN*/
    unsigned32        ClientID,    /*IN*/
    unsigned32        Type,         /*IN*/
    unsigned32        DeleteBoth,  /*IN*/
    ssm_fileset_id_t  FilesetID,   /*IN*/
    unsigned32        *RPCStatus); /*OUT*/
```

Description

The **ssm_FilesetDelete** function is used to delete DFS/HPSS and HPSS-only filesets.

Parameters

| | |
|-------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>DMGID</i> | Index into Server List for the DMG to be asked to delete the <i>fileset</i> . |
| <i>ClientID</i> | Index into the Client List. |
| <i>Type</i> | Type of fileset. Can be; <ul style="list-style-type: none"> SSM_DMG_FILESET_MO_C call dmg_admin_FilesetDelete. SSM_NS_FILESET_MO_C call hpss_FilesetDelete. SSM_NS_FILESET_FULL_MO_C call hpss_FilesetDelete. |
| <i>DeleteBoth</i> | Flag to delete the fileset from both DFS and HPSS. <ul style="list-style-type: none"> Used only when <i>Type</i> is SSM_DMG_FILESET_MO_C. See dmg_admin_FilesetDelete. |
| <i>FilesetID</i> | Contains the fileset name or id to be deleted. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_ID | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_BIND_FAIL | System Manager cannot bind to server. |
| SSM_ERPC_ERROR | RPC error. |

The return value from **dmg_admin_FilesetDelete** or **hpss_FilesetDelete**.

See also

ssm_FilesetDelete, **dmg_admin_FilesetDelete**, **hpss_FilesetDelete**.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_client_if.acf` file private to the calling application.

8.1.21. ssm_JobCancel**Purpose**

Cancel a job.

Syntax

```
#include "ssm_types.h"

signed32 ssm_JobCancel(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    unsigned32        ClientID,     /* IN */
    unsigned32        PVLID,        /* IN */
    job_id_t          JobID,        /* IN */
    unsigned32        *RPCStatus); /* OUT */
```

Description:

The ssm_JobCancel function allows an operator to cancel the specified job.

Parameters

| | |
|------------------|----------------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientID</i> | Index into Client List. |
| <i>PVLID</i> | PVL ID. |
| <i>JobID</i> | Job to cancel. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_EBIND_FAIL | System manager cannot bind to server. |
| SSM_ERPC_ERROR | RPC error. |

The return value from `w_pvl_DismountJobId`.

See also

None.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.22. ssm_JunctionCreate**Purpose**

Create a junction to a fileset.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_JunctionCreate (
    handle_t          Bh,          /*IN*/
    hpss_connect_handle_t *Ch,      /*IN*/
    unsigned32        ClientID,    /*IN*/
    char              *Path,        /*IN*/
    ssm_fileset_id_t  FilesetID,   /*IN*/
    unsigned32        *RPCStatus); /*OUT*/
```

Description

The **ssm_JunctionCreate** function is used to create a junction to an existing fileset.

Parameters

| | |
|------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientID</i> | Index into the Client List. |
| <i>Path</i> | Path to fileset. |
| <i>FilesetID</i> | Contains the fileset name or id to be deleted. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_ID | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_BIND_FAIL | System Manager cannot bind to server. |
| SSM_ERPC_ERROR | RPC error. |

The return value from **hpss_FilesetGetAttributes** or **hpss_JunctionCreate**.

See also

ssm_JunctionDelete, hpss_JunctionCreate.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.23. ssm_JunctionDelete**Purpose**

Delete a junction to a fileset.

Syntax

```
#include "ssm_types.h"

signed32 ssm_JunctionDelete (
    handle_t          Bh,          /*IN*/
    hpss_connect_handle_t *Ch,    /*IN*/
    unsigned32        ClientID,  /*IN*/
    char              *Path,      /*IN*/
    unsigned32        *RPCStatus); /*OUT*/
```

Description

The **ssm_JunctionDelete** function is used to delete a junction.

Parameters

| | |
|------------------|----------------------------------|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>ClientID</i> | Index into the Client List. |
| <i>Path</i> | Path to fileset. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_ID | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_BIND_FAIL | System Manager cannot bind to server. |
| SSM_ERPC_ERROR | RPC error. |

The return value from **hpss_JunctionDelete**.

See also

ssm_JunctionCreate, **hpss_JunctionDelete**.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.1.24. **ssm_ResourceCreate****Purpose**

Create resources in the Storage Server.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_ResourceCreate(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    unsigned32 ClientID,           /* IN */
    unsigned32 SSID,              /* IN */
    SsPVInfo_t *PVData,           /* IN */
    SsResources_t *SSData,        /* IN */
    unsigned32 *NumProcessed,      /* OUT */
    unsigned32 *RPCStatus);       /* OUT */
```

Description

The **ssm_ResourceCreate** function creates the physical volume, virtual volume, and storage map definitions necessary to make a set of volumes accessible by the storage system.

Parameters

| | | | | | | | |
|----------------------|--|-------------------|---------------------------|---------------|-----------------------------|----------------------|--------------------------|
| <i>Bh</i> | DCE binding handle. | | | | | | |
| <i>Ch</i> | HPSS connection handle. | | | | | | |
| <i>ClientID</i> | Index into Client List. | | | | | | |
| <i>SSID</i> | Storage server ID. | | | | | | |
| <i>PVData</i> | Identifies the list of physical volumes to add. It includes: <table> <tr> <td><i>PVName</i></td> <td>List of physical volumes.</td> </tr> <tr> <td><i>NumPVS</i></td> <td>Number of physical volumes.</td> </tr> </table> | <i>PVName</i> | List of physical volumes. | <i>NumPVS</i> | Number of physical volumes. | | |
| <i>PVName</i> | List of physical volumes. | | | | | | |
| <i>NumPVS</i> | Number of physical volumes. | | | | | | |
| <i>SSData</i> | Data for adding resources to Storage Server. It includes: <table> <tr> <td><i>VVSClassID</i></td> <td>Storage Class ID.</td> </tr> <tr> <td><i>Acct</i></td> <td>Account.</td> </tr> <tr> <td><i>EstimatedSize</i></td> <td>PV size (for disk only).</td> </tr> </table> | <i>VVSClassID</i> | Storage Class ID. | <i>Acct</i> | Account. | <i>EstimatedSize</i> | PV size (for disk only). |
| <i>VVSClassID</i> | Storage Class ID. | | | | | | |
| <i>Acct</i> | Account. | | | | | | |
| <i>EstimatedSize</i> | PV size (for disk only). | | | | | | |
| <i>NumProcessed</i> | Number of PV's processed. | | | | | | |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. | | | | | | |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |

| | |
|----------------------|--|
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| HPSS_ENOMEM | Memory allocation failure. |
| HPSS_ESYSTEM | Undetermined failure. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_ESS_INCONSISTENT | Storage server data is inconsistent (the PVs, VVs, and maps do not coordinate properly). |

The return code from **ss_PVCreate**, **ss_VVCreate**, **ss_MapCreate**, **ss_PVGetAttrs**, **ss_VVGetAttrs**, **ss_MapGetAttrs**.

See also

ssm_ResourceDelete.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_client_if.acf` file private to the calling application.

8.1.25. **ssm_ResourceDelete****Purpose**

Remove resources from the Storage Server.

Syntax

```
#include "ssm_types.h"

signed32 ssm_ResourceDelete(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    unsigned32 ClientID,           /* IN */
    unsigned32 SSID,              /* IN */
    SsPVInfo_t *PVData,           /* IN */
    unsigned32 *NumProcessed,      /* OUT */
    unsigned32 *RPCStatus);       /* OUT */
```

Description

The **ssm_ResourceDelete** function deletes the physical volume, virtual volume, and map definitions which define a set of volumes to the Storage Server, thus effectively deleting those volumes from the Storage Server.

Parameters

| | | | | | |
|---------------------|--|---------------|---------------------------|---------------|-----------------------------|
| <i>Bh</i> | DCE binding handle. | | | | |
| <i>Ch</i> | HPSS connection handle. | | | | |
| <i>ClientID</i> | Index into Client List. | | | | |
| <i>SSID</i> | SS ID. | | | | |
| <i>PVData</i> | Information with which to generate list of physical volumes. It includes: <table> <tr> <td><i>PVName</i></td> <td>List of physical volumes.</td> </tr> <tr> <td><i>NumPVS</i></td> <td>Number of physical volumes.</td> </tr> </table> | <i>PVName</i> | List of physical volumes. | <i>NumPVS</i> | Number of physical volumes. |
| <i>PVName</i> | List of physical volumes. | | | | |
| <i>NumPVS</i> | Number of physical volumes. | | | | |
| <i>NumProcessed</i> | Number of physical volumes processed. | | | | |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. | | | | |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|---|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |

HPSS_ECONFLICT Inconsistent VV data.

SSM_EMUTEX_FAIL Mutex failure.

SSM_ETRANABORT Transaction aborted.

The return code from **ss_VVGetAttrs**, **ss_MapDelete**, **ss_VVDelete**, **ss_PVDelete**.

See also

ssm_ResourceCreate.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_client_if.acf` file private to the calling application.

8.1.26. ssm_ResourceReclaim**Purpose**

Reclaim resources in a storage class.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_ResourceReclaim(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    unsigned32       ClientID,    /* IN */
    SsReclaim_t      *ReclaimData, /* IN */
    unsigned32       *RPCStatus); /* OUT */
```

Description

Reclaims resources in a storage class. Invokes the command line reclaim program.

Parameters

| | | | | | | | | | |
|---------------------------|---|-------------|-----------------|------------------|-----------------|-----------------|-------------------|---------------------------|--------------------|
| <i>Bh</i> | DCE binding handle. | | | | | | | | |
| <i>Ch</i> | HPSS connection handle. | | | | | | | | |
| <i>ClientID</i> | Index into Client List. | | | | | | | | |
| <i>ReclaimData</i> | Information for performing the reclaim, including: <table> <tr> <td><i>SSID</i></td> <td>Storage Server.</td> </tr> <tr> <td><i>NumberVVs</i></td> <td>VVs to reclaim.</td> </tr> <tr> <td><i>SClassID</i></td> <td>Storage class ID.</td> </tr> <tr> <td><i>WorkingDirectory[]</i></td> <td>Working directory.</td> </tr> </table> | <i>SSID</i> | Storage Server. | <i>NumberVVs</i> | VVs to reclaim. | <i>SClassID</i> | Storage class ID. | <i>WorkingDirectory[]</i> | Working directory. |
| <i>SSID</i> | Storage Server. | | | | | | | | |
| <i>NumberVVs</i> | VVs to reclaim. | | | | | | | | |
| <i>SClassID</i> | Storage class ID. | | | | | | | | |
| <i>WorkingDirectory[]</i> | Working directory. | | | | | | | | |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. | | | | | | | | |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|--|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation.. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_ECANT_MALLOC | Can't malloc. |
| SSM_EOPEN_FAIL | Can't open file for standard error. |
| SSM_ESM_INTERNAL_ERROR | Fork or exec failed. |

See also

`ssm_ResourceRepack`.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_client_if.acf` file private to the calling application.

8.1.27. ssm_ResourceRepack**Purpose**

Repacks resources in a storage class.

Syntax

```
#include "ssm_types.h"

signed32 ssm_ResourceRepack(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    unsigned32       ClientID,    /* IN */
    SsRepack_t       *RepackData, /* IN */
    unsigned32       *RPCStatus); /* OUT */
```

Description

Repacks resources in a storage class. Invokes the command line repack program.

The *SourceFile* and *TargetFile* fields of *RepackData* are currently ignored.

Parameters

| | | | | | | | | | | | | | | | |
|-------------------|--|-------------------|--|-------------------|--------------------------------------|-----------------|-------------------|------------------|--------------------------|-------------|-----------------|--------------|------------------------------|------------------|-----------------------------------|
| <i>Bh</i> | DCE binding handle. | | | | | | | | | | | | | | |
| <i>Ch</i> | HPSS connection handle. | | | | | | | | | | | | | | |
| <i>ClientID</i> | Index into Client List. | | | | | | | | | | | | | | |
| <i>RepackData</i> | Information for performing the repack, including: <table> <tr> <td><i>SourceFile</i></td> <td>File listing VVs from which to repack.</td> </tr> <tr> <td><i>TargetFile</i></td> <td>File listing VVs to which to repack.</td> </tr> <tr> <td><i>SClassID</i></td> <td>Storage Class ID.</td> </tr> <tr> <td><i>NumberVVs</i></td> <td>Number of VVs to repack.</td> </tr> <tr> <td><i>SSID</i></td> <td>Storage Server.</td> </tr> <tr> <td><i>Force</i></td> <td>Whether to force the repack.</td> </tr> <tr> <td><i>Threshold</i></td> <td>Space threshold at which to quit.</td> </tr> </table> | <i>SourceFile</i> | File listing VVs from which to repack. | <i>TargetFile</i> | File listing VVs to which to repack. | <i>SClassID</i> | Storage Class ID. | <i>NumberVVs</i> | Number of VVs to repack. | <i>SSID</i> | Storage Server. | <i>Force</i> | Whether to force the repack. | <i>Threshold</i> | Space threshold at which to quit. |
| <i>SourceFile</i> | File listing VVs from which to repack. | | | | | | | | | | | | | | |
| <i>TargetFile</i> | File listing VVs to which to repack. | | | | | | | | | | | | | | |
| <i>SClassID</i> | Storage Class ID. | | | | | | | | | | | | | | |
| <i>NumberVVs</i> | Number of VVs to repack. | | | | | | | | | | | | | | |
| <i>SSID</i> | Storage Server. | | | | | | | | | | | | | | |
| <i>Force</i> | Whether to force the repack. | | | | | | | | | | | | | | |
| <i>Threshold</i> | Space threshold at which to quit. | | | | | | | | | | | | | | |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. | | | | | | | | | | | | | | |

Return values

Upon successful completion the function returns 0. If an error occurs, the negated error code is returned.

Error conditions

| | |
|------------------------|--|
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EACCESS | The client does not have permission for this operation.. |
| SSM_EINVALID_CLIENT_ID | Invalid range for <i>ClientID</i> . |
| SSM_ECLIENT_NOT_REG | Client is not registered. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_ECANT_MALLOC | Can't malloc. |

SSM_EOPEN_FAIL Can't open file for standard error.

SSM_ESM_INTERNAL_ERROR
 Fork or exec failed.

See also

ssm_ResourceReclaim.

Clients

SSM Data Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_client_if.acf* file private to the calling application.

8.2. APIs Available to the Other HPSS Subsystems

The System Manager supplies an interface by which HPSS servers may notify it of significant events.

The Logging Subsystem forwards alarms, events, and status messages issued by the servers to the System Manager via the **ssm_LogMsgNotify** API. The PVR notifies the System Manager of pending tape mounts and tape mount completions by means of the **ssm_MountNotify** API.

The remaining APIs supported by this interface allow servers to forward notifications of changes in the attributes of a managed object.

8.2.1. `ssm_BitfileNotify`

Purpose

Receive notifications of changes to bitfile managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_BitfileNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    uuid_t           *SubsystemID, /* IN */
    u_signed64       SelectBitmap, /* IN */
    bfMO_attrib_t    *BitfileData, /* IN */
    unsigned32       *RPCStatus); /* OUT */
```

Description

The `ssm_BitfileNotify` function receives notifications of changes to a bitfile managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>BitfileData</i> | Latest copy of the complete bitfile managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Bitfile Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.2. `ssm_CartNotify`

Purpose

Receive notifications of changes to cartridge managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_CartNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    uuid_t             *SubsystemID, /* IN */
    u_signed64         SelectBitmap, /* IN */
    cart_data_t        *CartData,   /* IN */
    unsigned32         *RPCStatus); /* OUT */
```

Description

The `ssm_CartNotify` function receives notifications of changes to a cartridge managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>CartData</i> | Latest copy of the complete cartridge managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Physical Volume Repository.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.3. ssm_DeviceNotify

Purpose

Receive notifications of changes to device managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_DeviceNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    uuid_t            *SubsystemID, /* IN */
    u_signed64        *SelectBitmap, /* IN */
    devdesc_attr_t    *DeviceData,  /* IN */
    unsigned32        *RPCStatus); /* OUT */
```

Description

The **ssm_DeviceNotify** function receives notifications of changes to a device managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>DeviceData</i> | Latest copy of the complete device managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Mover.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.4. `ssm_DMGFilesetNotify`

Purpose

Receive notifications from a DMAP Gateway about fileset modifications.

Syntax

```
#include "ssm_types.h"

signed32 ssm_DMGFilesetNotify (
    handle_t          Bh,          /*IN*/
    hpss_connect_handle_t *Ch,    /*IN*/
    uuid_t           *SubsystemID, /*IN*/
    u_signed64       SelectBitmap, /*IN*/
    dmg_fileset_attr_t *FilesetData, /*IN*/
    unsigned32       *RPCStatus); /*OUT*/
```

Description

The `ssm_DMGFilesetNotify` function receives notifications of changes to a fileset managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>FilesetData</i> | Latest copy of the complete fileset managed object. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid SubsystemID. |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

DMAP Gateway

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.5. ssm_DMGNotify

Purpose

Receive notifications of changes to DMAP Gateway managed object.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_DMGNotify (  
    handle_t          Bh,          /*IN*/  
    hpss_connect_handle_t *Ch,      /*IN*/  
    uuid_t            *SubsystemID, /*IN*/  
    u_signed64        *SelectBitmap, /*IN*/  
    dmg_SpecificData_t *DMGData,    /*IN*/  
    unsigned32        *RPCStatus); /*OUT*/
```

Description

The **ssm_DMGFilesetNotify** function receives notifications of changes to a DMAP Gateway managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>DMGData</i> | Latest copy of the complete DMAP Gateway managed object. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|---------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

DMAP Gateway

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.6. `ssm_DriveNotify`

Purpose

Receive notifications of changes to drive managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_DriveNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    uuid_t            *SubsystemID, /* IN */
    u_signed64        SelectBitmap, /* IN */
    drive_data_t      *DriveData,  /* IN */
    unsigned32        *RPCStatus); /* OUT */
```

Description

The `ssm_DriveNotify` function receives notifications of changes to a drive managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>DriveData</i> | Latest copy of the complete drive managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Physical Volume Library.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.7. ssm_LogFileNotify

Purpose

Receive notifications of changes to logfile managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_LogFileNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    uuid_t            *SubsystemID, /* IN */
    u_signed64        SelectBitmap, /* IN */
    LogFileAttr_t     *LogFileData, /* IN */
    unsigned32        *RPCStatus); /* OUT */
```

Description

The **ssm_LogFileNotify** function receives notifications of changes to a logfile managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>LogFileData</i> | Latest copy of the complete logfile managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Log Daemon.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.8. `ssm_LogMsgNotify`

Purpose

Receive alarms, events, and status messages.

Syntax

```
#include "ssm_types.h"

signed32 ssm_LogMsgNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    uuid_t           SubsystemID, /* IN */
    log_msg_t       *LogMsgData, /* IN */
    unsigned32      *RPCStatus); /* OUT */
```

Description

The `ssm_LogMsgNotify` function receives alarms, events, and status messages which HPSS servers have sent to the logging subsystem. It passes them on to all Data Servers who are currently checked in.

Parameters

| | |
|--------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>LogMsgData</i> | An alarm, event, or status message. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to allocate the space to queue the log message.

Error conditions

| | |
|------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECANT_MALLOC | Can't malloc. |

See also

None.

Clients

Logging Client.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_server_if.acf` file private to the calling application.

8.2.9. `ssm_LSStatsNotify`

Purpose

Receive notifications of changes to Location Server statistics managed object.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_LSStatsNotify (
    handle_t           Bh,           /*IN*/
    hpss_connect_handle_t *Ch,       /*IN*/
    uuid_t            *SubsystemID, /*IN*/
    u_signed64        *SelectBitmap, /*IN*/
    ls_server_stats_t *StatsData,   /*IN*/
    unsigned32        *RPCStatus); /*OUT*/
```

Description

The `ssm_LSStatsNotify` function receives notifications of changes to a Location Server statistics managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>StatsData</i> | Latest copy of the complete Location Server statistics managed object. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Location Server

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.10. ssm_MPSNotify**Purpose**

Receive notifications of changes to MPS managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_MPSNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    uuid_t           *SubsystemID, /* IN */
    u_signed64       SelectBitmap, /* IN */
    mps_attrib_t     *MPSPData,   /* IN */
    unsigned32       *RPCStatus); /* OUT */
```

Description

The **ssm_MPSNotify** function receives notifications of changes to a MPS managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>MPSPData</i> | Latest copy of the complete MPS managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

MPS.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.11. ssm_MPS_SClassNotify**Purpose**

Receive notifications of changes to storage class managed object.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_MPS_SClassNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    uuid_t           *SubsystemID, /* IN */
    u_signed64       SelectBitmap, /* IN */
    mps_sclass_attrs_t *SClassData, /* IN */
    unsigned32       *RPCStatus);  /* OUT */
```

Description

The **ssm_MPS_SClassNotify** function receives notifications of changes to a storage class managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>SClassData</i> | Latest copy of the complete storage class managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

MPS.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.12. ssm_MVRNotify**Purpose**

Receive notifications of changes to Mover managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_MVRNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    uuid_t             *SubsystemID, /* IN */
    u_signed64         *SelectBitmap, /* IN */
    mover_attr_t       *MVRData,     /* IN */
    unsigned32         *RPCStatus); /* OUT */
```

Description

The **ssm_MVRNotify** function receives notifications of changes to a Mover managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>MVRData</i> | Latest copy of the complete Mover managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Mover.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.13. ssm_MapNotify**Purpose**

Receive notifications of changes to storage map managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_MapNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    uuid_t            *SubsystemID, /* IN */
    u_signed64        SelectBitmap, /* IN */
    ss_map_attr_t     *MapData,     /* IN */
    unsigned32        *RPCStatus);  /* OUT */
```

Description

The **ssm_MapNotify** function receives notifications of changes to a storage map managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>MapData</i> | Latest copy of the complete storage map managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Storage Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.14. ssm_MountNotify**Purpose**

Receive notifications of pending or completed mounts.

Syntax

```
signed32 ssm_MountNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    uuid_t             *PVRID,     /* IN */
    cart_t             *Cart,      /* IN */
    side_t             *Side,      /* IN */
    drive_addr_t       *Drive,     /* IN */
    unsigned32         RequestType, /* IN */
    unsigned32         *RPCStatus); /* OUT */
```

Description

The **ssm_MountNotify** function receives notifications that a tape mount has been requested or completed. It passes the notification on to all Data Server processes which are currently checked in.

Parameters

| | |
|--------------------|--|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>PVRID</i> | PVR ID. |
| <i>Cart</i> | Cartridge ID. |
| <i>Side</i> | Side; null if the cartridge has only one side. |
| <i>Drive</i> | Drive; null if no specific drive is requested. |
| <i>RequestType</i> | Request type (mount requested or mount completed). |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|--------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_ESRV_NOT_FOUND | Server not found. |

See also

None.

Clients

Physical Volume Repository.

Notes

None.

8.2.15. ssm_NFS2_StatsNotify**Purpose**

Receive notifications of changes to NFS statistics managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_NFS2_StatsNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    uuid_t            *SubsystemID, /* IN */
    u_signed64        SelectBitmap, /* IN */
    nfs2_stats_t      *StatsData,   /* IN */
    unsigned32        *RPCStatus);  /* OUT */
```

Description

The **ssm_NFS2_StatsNotify** function receives notifications of changes to a NFS statistics managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>StatsData</i> | Latest copy of the complete NFS statistics managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |

SSM_ESRV_NOT_FOUND Server with *SubsystemID* not found.

See also

None.

Clients

NFS Daemon.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.16. ssm_NSFilesetNotify**Purpose**

Receive notifications of changes to Name Server fileset managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_NSFilesetNotify (
    handle_t           Bh,           /*IN*/
    hpss_connect_handle_t *Ch,       /*IN*/
    uuid_t             *SubsystemID, /*IN*/
    u_signed64         *SelectBitmap, /*IN*/
    ns_FilesetAttrs_t *FilesetData,  /*IN*/
    unsigned32         *RPCStatus); /*OUT*/
```

Description

The **ssm_NSFilesetNotify** function receives notifications of changes to a Name Server fileset managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| SubsystemID | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>FilesetData</i> | Latest copy of the complete Name Server fileset managed object. |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Name Server

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.17. ssm_NSNotify**Purpose**

Receive notifications of changes to Name Server managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_NSNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    uuid_t             *SubsystemID, /* IN */
    u_signed64         SelectBitmap, /* IN */
    ns_SpecificConfig_t *NSData,     /* IN */
    unsigned32         *RPCStatus); /* OUT */
```

Description

The **ssm_NSNotify** function receives notifications of changes to a Name Server managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>NSData</i> | Latest copy of the complete Name Server managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Name Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.18. ssm_PVLNotify**Purpose**

Receive notifications of changes to PVL managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_PVLNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    uuid_t            *SubsystemID, /* IN */
    u_signed64        SelectBitmap, /* IN */
    pvl_data_t        *PVLData,    /* IN */
    unsigned32        *RPCStatus);  /* OUT */
```

Description

The **ssm_PVLNotify** function receives notifications of changes to a PVL managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>PVLData</i> | Latest copy of the complete PVL managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Physical Volume Library.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.19. ssm_PVNotify**Purpose**

Receive notifications of changes to Storage Server physical volume managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_PVNotify(
    handle_t                Bh,           /* IN */
    hpss_connect_handle_t  *Ch,         /* IN */
    uuid_t                  *SubsystemID, /* IN */
    u_signed64              SelectBitmap, /* IN */
    pv_attr_t               *PVData,     /* IN */
    unsigned32              *RPCStatus); /* OUT */
```

Description

The **ssm_PVNotify** function receives notifications of changes to a physical volume managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>PVData</i> | Latest copy of the complete physical volume managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Storage Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.20. ssm_PVRNotify**Purpose**

Receive notifications of changes to PVR managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_PVRNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    uuid_t            *SubsystemID, /* IN */
    u_signed64        SelectBitmap, /* IN */
    pvr_data_t        *PVRData,     /* IN */
    unsigned32        *RPCStatus); /* OUT */
```

Description

The **ssm_PVRNotify** function receives notifications of changes to a PVR managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>PVRData</i> | Latest copy of the complete PVR managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Physical Volume Repository.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.21. ssm_QueueNotify**Purpose**

Receive notifications of changes to PVL queue managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_QueueNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    uuid_t             *SubsystemID, /* IN */
    u_signed64         SelectBitmap, /* IN */
    api_queue_data_t   *QueueData,   /* IN */
    unsigned32         *RPCStatus); /* OUT */
```

Description

The **ssm_QueueNotify** function receives notifications of changes to a PVL queue managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>QueueData</i> | Latest copy of the complete PVL queue managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Physical Volume Library.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.22. ssm_RequestNotify**Purpose**

Receive notifications of changes to PVL request managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_RequestNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    uuid_t           *SubsystemID, /* IN */
    u_signed64       SelectBitmap, /* IN */
    request_data_t   *RequestData, /* IN */
    unsigned32       *RPCStatus);  /* OUT */
```

Description

The **ssm_RequestNotify** function receives notifications of changes to a PVL request managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>RequestData</i> | Latest copy of the complete PVL request managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Physical Volume Library.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.23. ssm_SFSNotify**Purpose**

Receive notifications of changes to SFS managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_SFSNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    uuid_t           *SubsystemID, /* IN */
    u_signed64       SelectBitmap, /* IN */
    sfs_attrs_t      *SFSDData,   /* IN */
    unsigned32       *RPCStatus);  /* OUT */
```

Description

The **ssm_SFSNotify** function receives notifications of changes to the SFS managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>SFSDData</i> | Latest copy of the complete SFS managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Metadata Monitor.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.24. ssm_SSNotify**Purpose**

Receive notifications of changes to storage segment managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_SSNotify(
    handle_t                Bh,                /* IN */
    hpss_connect_handle_t  *Ch,                /* IN */
    uuid_t                  *SubsystemID,      /* IN */
    u_signed64              SelectBitmap,      /* IN */
    ss_attr_t               *SSData,          /* IN */
    unsigned32              *RPCStatus);      /* OUT */
```

Description

The **ssm_SSNotify** function receives notifications of changes to a storage segment managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>SSData</i> | Latest copy of the complete storage segment managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Storage Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.25. ssm_ServerNotify**Purpose**

Receive notifications of changes to server managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_ServerNotify(
    handle_t                Bh,           /* IN */
    hpss_connect_handle_t  *Ch,         /* IN */
    uuid_t                  *SubsystemID, /* IN */
    u_signed64              SelectBitmap, /* IN */
    hpss_server_attr_t      *ServerData, /* IN */
    unsigned32              *RPCStatus); /* OUT */
```

Description

The **ssm_ServerNotify** function receives notifications of changes to a server managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>ServerData</i> | Latest copy of the complete server managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Bitfile Server, Logging Client, Logging Daemon, Metadata Monitor, MPS, Mover, Mount Daemon, NFS Daemon, Name Server, Physical Volume Library, Physical Volume Repository, Storage Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an `ssm_server_if.acf` file private to the calling application.

8.2.26. ssm_SsrvNotify**Purpose**

Receive notifications of changes to Storage Server managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_SsrvNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,    /* IN */
    uuid_t           *SubsystemID, /* IN */
    u_signed64       SelectBitmap, /* IN */
    ssrv_attr_t      *SsrvData,    /* IN */
    unsigned32       *RPCStatus);  /* OUT */
```

Description

The **ssm_SsrvNotify** function receives notifications of changes to a Storage Server managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>SsrvData</i> | Latest copy of the complete Storage Server managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Storage Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.27. ssm_TapeCheckInNotify**Purpose**

Receive notifications of tape check-in requests from the PVR.

Syntax

```
signed32 ssm_TapeCheckInNotify(
    handle_t          Bh,          /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    uuid_t            *PVRID,     /* IN */
    cart_t            *Cart,       /* IN */
    side_t            *Side,       /* IN */
    ioport_addr_t     *IOPort,    /* IN */
    unsigned32        RequestType, /* IN */
    unsigned32        *RPCStatus); /* OUT */
```

Description

The **ssm_TapeCheckInNotify** function receives notifications that a tape check-in has been requested. It passes the notification on to all Data Server processes which are currently checked in.

Parameters

| | |
|--------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>PVRID</i> | PVR ID. |
| <i>Cart</i> | Cartridge ID. |
| <i>Side</i> | Side; null if the cartridge has only one side. |
| <i>IOPort</i> | I/O Port where the cartridge is to be inserted. |
| <i>RequestType</i> | Request type (insert cartridge or cartridge has been inserted). |
| <i>RPCStatus</i> | RPC error code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|--------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_EINVALID_IN | Invalid input. |
| SSM_ESRV_NOT_FOUND | Server not found. |

See also

None.

Clients

Physical Volume Repository.

Notes

None.

8.2.27. ssm_VVNotify**Purpose**

Receive notifications of changes to virtual volume managed object.

Syntax

```
#include "ssm_types.h"
```

```
signed32 ssm_VVNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,      /* IN */
    uuid_t             *SubsystemID, /* IN */
    u_signed64         SelectBitmap, /* IN */
    vv_attr_t          *VVData,     /* IN */
    unsigned32         *RPCStatus);  /* OUT */
```

Description

The **ssm_VVNotify** function receives notifications of changes to a virtual volume managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>VVData</i> | Latest copy of the complete virtual volume managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Storage Server.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.2.28. ssm_VolNotify**Purpose**

Receive notifications of changes to PVL volume managed object.

Syntax

```
#include "ssm_types.h"

signed32 ssm_VolNotify(
    handle_t           Bh,           /* IN */
    hpss_connect_handle_t *Ch,       /* IN */
    uuid_t             *SubsystemID, /* IN */
    u_signed64         SelectBitmap, /* IN */
    vol_data_t         *VolData,     /* IN */
    unsigned32         *RPCStatus); /* OUT */
```

Description

The **ssm_VolNotify** function receives notifications of changes to a PVL volume managed object and passes them on to Data Servers who are registered to receive them.

Parameters

| | |
|---------------------|---|
| <i>Bh</i> | RPC binding handle. |
| <i>Ch</i> | HPSS connection handle. |
| <i>SubsystemID</i> | UUID of the server sending the notification. |
| <i>SelectBitmap</i> | Bitmap showing which attributes of the managed object have changed. |
| <i>VolData</i> | Latest copy of the complete PVL volume managed object. |
| <i>RPCStatus</i> | RPC return code, supplied by DCE. |

Return values

Upon successful completion the function returns 0. Errors indicate either a bad connection or an internal failure of the System Manager to lock or access its table of registered managed objects.

Error conditions

| | |
|------------------------|---|
| HPSS_EBADCONN | Bad connection handle. |
| HPSS_ENOTREADY | The System Manager has not yet initialized. |
| SSM_ECONDITION_FAIL | Condition variable failure. |
| SSM_ECONDITION_TIMEOUT | Condition variable time-out. |
| SSM_EMUTEX_FAIL | Mutex failure. |
| SSM_EINVALID_IN | Invalid <i>SubsystemID</i> . |
| SSM_ESRV_NOT_FOUND | Server with <i>SubsystemID</i> not found. |

See also

None.

Clients

Physical Volume Library.

Notes

The *RPCStatus* parameter is optional. If it is used, it must be declared for the function in an *ssm_server_if.acf* file private to the calling application.

8.3. Data Definitions

8.3.1. Data Common to the System Manager and the Data Server

8.3.1.1. Significant constants

| | |
|--------------------|--|
| SSM_MAX_CLIENT | The maximum number of clients who may check in concurrently. |
| SSM_NEW_CLIENT | This constant is passed as the input client ID by the Data Server at its first check-in. It must be equal to SSM_MAX_CLIENTS. |
| SSM_SEC_SERVER_ID | The "server ID" of the security services. This is a convenient way to request information about the security service without actually pretending it is a "server". It must always be greater than SSM_MAX_SERVERS. |
| SSM_SSM_CLIENT_ID | This is a convenient way to let SSM be a "client" to itself. It is set to something greater than SSM_MAX_CLIENTS. |
| LIST STATUS VALUES | The various lists shared by the System Manager and Data Server may have statuses with the following values: |
| SSM_VALID | The list is valid. |
| SSM_INVALID | There is some inconsistency in the list. |
| SSM_MMFAIL | There was an error opening or reading the metadata for one or more entries. |

8.3.1.2. Server List - ServerList_t

Description

The Server List is a list of the servers defined in the HPSS Server Configuration File. The System Manager builds the list at startup time and modifies it whenever a server is added, deleted, or updated.

The index of a server in the Server List is its primary means of identification for the System Manager and Data Server. The System Manager and Data Server call this index the "Server ID". We are aware that most of HPSS uses the term "Server ID" to mean the server's uuid, but the term has stuck, anyway. We try to specify "Server UUID" whenever we are talking about the uuid.

Slots in the Server List are never reused during one instantiation of the System Manager. If a server is removed from the HPSS Server Configuration File, its SSM_SVRFLAG_INUSE Flag bit is cleared. This is because SSM relies so heavily on the Server ID to identify the server and the windows which may be open for it. If the System Manager is restarted, the servers will be reordered without any holes in the list. The "ID" column in the Server List Window specifies the Server ID.

Format

```
typedef struct ServerStruct {
    unsigned32    Flags;
    unsigned32    Type;
    unsigned32    Subtype;
    unsigned32    Opstate;
    [string] char DescName[HPSS_MAX_DESC_NAME];
    [string] char  Host[HPSS_MAX_HOST_NAME];
}
```

```
    [string] char    HpssdName[HPSS_MAX_HOST_NAME];  
    uuid_t          UUID;  
} ServerStruct_t;
```

Flags

Indicates characteristics and states of server.

Type

Server type.

Subtype

Server subtype.

OpState

Server operational state.

DescName

Server descriptive name.

Host

Server host on which the server is executing.

HpssdName

The hostname of the startup daemon that controls this server.

UUID

Server UUID.

```
typedef struct ServerList {  
    unsigned32          Size;  
    unsigned32          Max;  
    unsigned32          ChunkSize;  
    unsigned32          Status;  
    unsigned32          Version;  
    timestamp_t         CreateTime;  
    timestamp_t         UpdateTime;  
    [size_is(Size)] ServerStruct_t List[*];  
} ServerList_t;
```

Size

Number of servers in the list.

Max

Maximum number of servers in the list.

ChunkSize

Number of servers to add to the list when the current list becomes full.

Status

List status.

Version

Changes when the list changes.

CreateTime

Time that the list was first created.

UpdateTime

Time that the list was last updated.

List

List of servers.

Valid values for Flags:

| | |
|-----------------------|--|
| SSM_SVRFLAG_INUSE | A server has been assigned to this slot in the Server List. |
| SSM_SVRFLAG_UP | The server has been confirmed to be up. |
| SSM_SVRFLAG_DOWN | The server has been confirmed to be down. |
| SSM_SVRFLAG_CONNECTED | The System Manager is currently connected to the server. |
| SSM_SVRFLAG_STARTABLE | The server can be started by the System Manager via the startup daemon. The servers which are not STARTABLE are the System Manager itself and the startup daemon, which must be started manually. |
| SSM_SVRFLAG_EXEC | The EXECUTE_SERVER_FLAG for this server in the General Server Configuration file is ON. |
| SSM_SVRFLAG_ADMIN | The server has a server managed object and allows SSM to set its Administrative State. |
| SSM_SVRFLAG_SNOOTY | The server has no DCE interface to SSM. This flag is losing its usefulness as we no longer keep SNOOTY servers, such as ftpd or pfsd, in the Server Configuration file. |
| SSM_SVRFLAG_DELETED | The server has been deleted from the General Server Configuration file. It may or may not still be executing. |
| SSM_SVRFLAG_EAUTH | SSM is not authorized to connect to the server. If neither the UP nor the DOWN flag is set, it means that we can't connect to the server but we can't talk to his startup demon, either, so we don't know for certain whether he is up or not. The Data Server will display a Status of INDETERMINATE for such servers. The UP and DOWN flags should never both be set at the same time. |
| SSM_SVRFLAG_CONFIG | The SM has determined that there is some type of a configuration problem with the server. This normally means that an SFS file that is referenced can not be |

found or that an entry in an SFS file is non-existent. The Data Server will display a Status of CHECK CONFIG for these servers.

The previous states are set only by the System Manager. The following values may be set by either the System Manager or the Data Server, but the System Manager may override any state set by the Data Server. They are considered transient states.

| | |
|------------------------|---|
| SSM_SVRFLAG_STARTING | SSM is trying to start the server. |
| SSM_SVRFLAG_STOPPING | SSM is trying to shut down the server. |
| SSM_SVRFLAG_HALTING | SSM is trying to halt the server. |
| SSM_SVRFLAG_REINITING | SSM is trying to reinitialize the server. |
| SSM_SVRFLAG_REPAIRING | SSM is trying to repair the server. |
| SSM_SVRFLAG_CONNECTING | SSM is trying to connect to the server. |

The following states are set only by the Data Server for its private use.

SSM_SVRFLAG_CLIENT1
SSM_SVRFLAG_CLIENT2

Valid values for Status:

The list status values SSM_VALID, SSM_INVALID and SSM_MMFAIL may all be applied to the Server List. In addition, the Server List Status may be:

| | |
|------------------------|---|
| SSM_EXPECTED_NOT_FOUND | There was no entry of type ssm in the server configuration file which had a descriptive name matching the value of the HPSS_DESC_SSMSM environment variable. The System Manager will make up an appropriate entry for the server list and try to add it to the server configuration file. |
| SSM_ADDED_SSM | The SSM_EXPECTED_NOT_FOUND condition is true, and the System Manager was successful in adding its made-up entry to the server configuration file. |
| SSM_UNEXPECTED_FOUND | There were one or more entries of type ssm in the server configuration file which had a descriptive name which did not match the HPSS_DESC_SSMSM environment variable. |
| SSM_CDS_WRONG | There was an entry of type ssm in the server configuration file which had a descriptive name which matched the value of the HPSS_DESC_SSMSM environment variable, but its ServerName field did not match the value of the HPSS_CDS_SSMSM environment variable. |

Clients

The System Manager passes the Server List to the Data Server at check-in. The System Manager sends the Data Server a new copy of the Server List in a notification whenever the list changes. Since the Server List can change so often, a special thread monitors it and sends the Data Server a new copy only at periodic (very small) intervals, rather than every single time the list changes.

8.3.1.3. Drive List - DriveList_t

Description

The Drive List is constructed by the System Manager at startup and updated whenever a drive is added or deleted from the PVL Drive Configuration File and the Mover Device Configuration File.

Format

```
typedef struct DriveStruct {
    unsigned32    DeviceID;
    unsigned32    Flags;
    [string] char MvrDeviceName[MVR_MAXDEVNAME];
    unsigned32    MvrID;
    [string] char MvrHost[HPSS_MAX_HOST_NAME];
    drive_addr_t PvlDriveAddress;
    unsigned32    PvlID;
    unsigned32    PvrID;
} DriveStruct_t;
```

DeviceID

Common device/drive ID.

Flags

Characteristics and states of drive.

MvrDeviceName

Mover device name.

MvrID

Index of Mover in Server List.

PvlDriveAddress

PVL drive address.

PvlID

Index of PVL in Server List.

PvrID

Index of PVR in Server List.

```
typedef struct DriveList {
    unsigned32    Size;
    unsigned32    Max;
    unsigned32    ChunkSize;
    unsigned32    Status;
    unsigned32    Version;
    [size_is(Size)] DriveStruct_t List[*];
} DriveList_t;
```

Size

Number of device/drives in list.

Max

Maximum number of device/drives in list.

ChunkSize

Number of device/drives to add to the list when the current list becomes full..

Status

List status.

Version

Changes when list changes.

List

List of drives.

Valid values for Flags:

| | |
|--------------------------------|---|
| SSM_DRIVELIST_NODEV | There is no corresponding device for a defined drive. |
| SSM_DRIVELIST_NODRIVE | There is no corresponding drive for a defined device. |
| SSM_DRIVELIST_BAD_MVR_UUID_DEV | There is an unknown Mover uuid in a device file. |
| SSM_DRIVELIST_BAD_MVR_UUID_DRV | There is an unknown Mover uuid in the drive file. |
| SSM_DRIVELIST_BAD_PVR_UUID | There is an unknown PVR uuid for a drive. |
| SSM_DRIVELIST_MVR_MISMATCH | The Mover specified for a device in the device file is not same as the Mover specified for the same device in the drive file. |

Clients

The Drive List is passed to the Data Server at check-in to enable the Data Server to construct selection lists on drives and provide the proper identifying information on drives in subsequent configuration calls. The System Manager sends the Data Server a new copy of the Drive List in a notification whenever the list changes.

8.3.1.4 Class of Service List - COSList_t

Description

The Class of Service List is a list of classes of service defined in the COS Configuration File. The System Manager builds the list at startup time and modifies it whenever a class of service is added, deleted, or updated.

Format

```
typedef struct COSStruct {  
    cos_t          ID;
```

```
[string] char      Name[HPSS_MAX_COS_NAME];
} COSStruct_t;
```

ID

Class of Service ID.

Name

Class of Service name.

```
typedef struct COSList {
    unsigned32      Size;
    unsigned32      Status;
    unsigned32      Version;
    [size_is(Size)] COSStruct_t List[*];
} COSList_t;
```

Size

Number of entries defined in the list.

Status

List status.

Version

Changes when the list changes.

List

List of entries.

Clients

The System Manager passes the Class of Service List to the Data Server at check-in so that the Data Server can build a selection list for classes of service. The System Manager sends the Data Server a new copy of the Class of Service List in a notification whenever the list changes.

8.3.1.5. Storage Class List - SClassList_t**Description**

The Storage Class List is a list of classes of service defined in the Storage Class Configuration File. The System Manager builds the list at startup time and modifies it whenever a storage class is added, deleted, or updated.

Format

```
typedef struct SClassList {
    unsigned32      Size;
    unsigned32      Status;
    unsigned32      Version;
    [size_is(Size)] hpss_sclass_md_t List[*];
} SClassList_t;
```

Size

Number of entries defined in the list.

Status

List status.

Version

Changes when the list changes.

List

List of entries.

Clients

The System Manager passes the Storage Class List to the Data Server at check-in so that the Data Server can build a selection list for storage classes and manage the storage class list window. The System Manager sends the Data Server a new copy of the Storage Class List in a notification whenever the list changes.

8.3.1.6. Hierarchy List - HierList_t

Description

The Hierarchy List is a list of hierarchies defined in the Hierarchy Configuration File. The System Manager builds the list at startup time and modifies it whenever a hierarchy is added, deleted, or updated.

Format

```
typedef struct HierStruct {
    unsigned32      ID;
    [string]char    Name[HPSS_MAX_OBJECT_NAME];
} HierStruct_t;
```

ID

Hierarchy ID.

Name

Hierarchy name.

```
typedef struct HierList {
    unsigned32      Size;
    unsigned32      Status;
    unsigned32      Version;
    [size_is(Size)] HierStruct_t List[*];
} HierList_t;
```

Size

Number of entries defined in the list.

Status

List status.

Version

Changes when the list changes.

List

List of entries.

Clients

The System Manager passes the Hierarchy List to the Data Server at check-in so that the Data Server can

build a selection list for hierarchies. The System Manager sends the Data Server a new copy of the Hierarchy List in a notification whenever the list changes.

8.3.1.7. Migration Policy List - MigrPList_t

Description

The Migration Policy List is a list of migration policies defined in the Migration Policy Configuration File. The System Manager builds the list at startup time and modifies it whenever a migration policy is added, deleted, or updated.

Format

```
typedef struct MigrPStruct {
    unsigned32      ID;
    [string] char   Name[HPSS_MAX_OBJECT_NAME];
} MigrPStruct_t;
```

ID

Migration Policy ID.

Name

Migration Policy name.

```
typedef struct MigrPList {
    unsigned32      Size;
    unsigned32      Status;
    unsigned32      Version;
    [size_is(Size)] MigrPStruct_t List[*];
} MigrPList_t;
```

Size

Number of entries defined in the list.

Status

List status.

Version

Changes when the list changes.

List

List of entries.

Clients

The System Manager passes the Migration Policy List to the Data Server at check-in so that the Data Server can build a selection list for classes of service. The System Manager sends the Data Server a new copy of the Migration Policy List in a notification whenever the list changes.

8.3.1.8. Purge Policy List - PurgPList_t

Description

The Purge Policy List is a list of purge policies defined in the Purge Policy Configuration File. The System Manager builds the list at startup time and modifies it whenever a purge policy is added, deleted, or

updated.

Format

```
typedef struct PurgPStruct {
    unsigned32      ID;
    [string] char   Name[HPSS_MAX_OBJECT_NAME];
} PurgPStruct_t;
```

ID

Purge Policy ID.

Name

Purge Policy name.

```
typedef struct PurgPList {
    unsigned32      Size;
    unsigned32      Status;
    unsigned32      Version;
    [size_is(Size)] PurgPStruct_t List[*];
} PurgPList_t;
```

Size

Number of entries defined in the list.

Status

List status.

Version

Changes when the list changes.

List

List of entries.

Clients

The System Manager passes the Purge Policy List to the Data Server at check-in so that the Data Server can build a selection list for purge policies. The System Manager sends the Data Server a new copy of the Purge Policy List in a notification whenever the list changes.

8.3.1.9. Notification Structure - NotifyUnion_t

The notifications sent by the System Manager to the Data Server use the Client Notification Structure NotifyUnion_t, which is defined in the *Other Data Definitions (Data Server Clients)* section later in this chapter.

8.3.1.10. Server Info Data Structure - SrvInfoUnion_t

Description

The Server Info Data structure is a union of all data types passed between the System Manager and Data Server for managed objects, configuration file entries, and object ids.

Format

The pointer default class is type full.

```
typedef union switch(CFClass_enum_t SrvInfoClass) SrvInfoData {

    case SSM_INVALID_MO_C:                unsigned32                *InvalidMO;
    case SSM_ALL_FILE_MO_C:               ssm_fileattr_t           *AllFileMO;
    case SSM_ALL_SERVER_MO_C:             hpss_server_attr_t       *AllServerMO;
    case SSM_BFS_BITFILE_MO_C:            bfMO_attr_t              *BfsBitfileMO;
    case SSM_DMG_DMG_MO_C:                 dmg_SpecificData_t       *DmgDmgMO;
    case SSM_DMG_FILESET_MO_C:            dmg_fileset_attr_t       *DmgFilesetMO;
    case SSM_DMG_FILESET_FULL_MO_C:       ssm_dmg_fileset_t        *DmgFilesetFullMO;
    case SSM_DMG_FILESET_LIST_MO_C:       dmg_FSConfArray_t        *DmgFilesetListMO;
    case SSM_HDM_FILESET_MO_C:            dmg_hdm_ex_fileset_info_t *HdmFilesetMO;
    case SSM_LOGD_LOGFILE_MO_C:           LogFileAttr_t            *LogdLogfileMO;
    case SSM_LS_STATS_MO_C:               ls_server_stats_t        *LsStatsMO;
    case SSM_MM_SFS_MO_C:                 sfs_attrs_t              *MmSfsMO;
    case SSM_MPS_MPS_MO_C:                 mps_attr_t               *MpsMpsMO;
    case SSM_MPS_SCLASS_MO_C:             mps_sclass_attr_t        *MpsSClassMO;
    case SSM_MPS_SCLASS_ALL_MO_C:         mps_sclass_attr_array_t  *MpsSClassAllMO;
    case SSM_MVR_DEVICE_MO_C:             devdesc_attr_t           *MvrDeviceMO;
    case SSM_MVR_MVR_MO_C:               mover_attr_t             *MvrMvrMO;
    case SSM_NFS2_STATS_MO_C:             nfs2_stats_t             *Nfs2StatsMO;
    case SSM_NS_NS_MO_C:                 ns_SpecificConfig_t      *NsNsMO;
    case SSM_NS_FILESET_MO_C:             ns_FilesetAttrs_t        *NsFilesetMO;
    case SSM_NS_FILESET_FULL_MO_C:        ssm_ns_fileset_t         *NsFilesetFullMO;
    case SSM_PVL_DRIVE_MO_C:              drive_data_t             *PvlDriveMO;
    case SSM_PVL_PVL_MO_C:               pvl_data_t               *PvlPvlMO;
    case SSM_PVL_QUEUE_MO_C:             api_queue_data_t         *PvlQueueMO;
    case SSM_PVL_REQUEST_MO_C:           request_data_t           *PvlRequestMO;
    case SSM_PVL_VOL_MO_C:               vol_data_t               *PvlVolMO;
    case SSM_PVR_CART_MO_C:              cart_data_t              *PvrCartMO;
    case SSM_PVR_PVR_MO_C:               pvr_data_t               *PvrPvrMO;
    case SSM_SS_MAP_MO_C:                 ss_map_attr_t            *SsMapMO;
    case SSM_SS_PV_MO_C:                 pv_attr_t                 *SsPvMO;
    case SSM_SS_SS_MO_C:                 ss_attr_t                 *SsSsMO;
    case SSM_SS_SSRV_MO_C:               ssrv_attr_t              *SsSsrvMO;
    case SSM_SS_VV_MO_C:                 vv_attr_t                 *SsVvMO;

    case SSM_ALL_ACCT_CF_C:               acct_config_t             *AllAcctCF;
    case SSM_ALL_COS_CF_C:               hpss_cos_md_t            *AllCosCF;
    case SSM_ALL_DEV_DRIVE_CF_C:         DriveData_t              *AllDevDriveCF;
    case SSM_ALL_FILE_FAMILY_CF_C:       hpss_file_family_md_t    *AllFileFamilyCF;
    case SSM_ALL_HIER_CF_C:              hpss_hier_md_t           *AllHierCF;
    case SSM_ALL_LS_CF_C:                ls_policy_md_t           *AllLsCF;
    case SSM_ALL_REMOTE_SITE_CF_C:       hpss_site_md_t           *AllRemoteSiteCF;
    case SSM_ALL_MIGR_POLICY_CF_C:       hpss_migr_policy_md_t    *AllMigrPolicyCF;
    case SSM_ALL_PURG_POLICY_CF_C:       hpss_purge_policy_md_t   *AllPurgPolicyCF;
    case SSM_ALL_SCLASS_CF_C:            hpss_sclass_md_t         *AllSClassCF;
    case SSM_ALL_SERVER_CF_C:            hpss_server_config_t     *AllServerCF;
    case SSM_BFS_BFS_CF_C:               bfs_config_info_t        *BfsBfsCF;
    case SSM_BFS_BFSEGDISK_CF_C:         bf_disk_segment_md_t     *BfsBfsegdiskCF;
    case SSM_BFS_BFSEGTAPE_CF_C:         bf_tape_segment_md_t     *BfsBfsegtapeCF;
    case SSM_BFS_BITFILE_CF_C:           bf_descriptor_md_t       *BfsBitfileCF;
    case SSM_DMG_DMG_CF_C:               dmg_gateway_config_t     *DmgDmgCF;
    case SSM_DMG_FILESET_CF_C:           dmg_fileset_info_t       *DmgFilesetCF;
    case SSM_LOGC_LOGC_CF_C:             LogcConfig_t             *LogcLogcCF;
    case SSM_LOGC_POLICY_CF_C:           LogcPolicy_t             *LogcPolicyCF;
    case SSM_LOGD_LOGD_CF_C:             LogdConfig_t             *LogdLogdCF;
    case SSM_MM_MM_CF_C:                 mm_mon_config_t         *MmMmCF;
    case SSM_MNT1_MNT1_CF_C:             mountd_config_t          *Mnt1Mnt1CF;
    case SSM_MPS_MPS_CF_C:               mps_config_t             *MpsMpsCF;
    case SSM_MVR_DEVICE_CF_C:            device_desc_md_t         *MvrDeviceCF;
    case SSM_MVR_MVR_CF_C:               mvr_config_t             *MvrMvrCF;
    case SSM_NFS2_NFS2_CF_C:             nfs2_config_t            *Nfs2Nfs2CF;
    case SSM_NS_NS_CF_C:                 ns_SpecificConfig_t      *NsNsCF;

```

```
case SSM_PVL_DRIVE_CF_C:      drive_data_t      *PvlDriveCF;
case SSM_PVL_PVL_CF_C:       pvl_data_t        *PvlPvlCF;
case SSM_PVL_VOL_CF_C:       vol_data_t         *PvlVolCF;
case SSM_PVR_CART_CF_C:      cart_data_t         *PvrCartCF;
case SSM_PVR_PVR_CF_C:      pvr_data_t         *PvrPvrCF;
case SSM_SS_MAP_CF_C:        storage_map_md_t    *SsMapCF;
case SSM_SS_PV_CF_C:         physical_volume_md_t  *SsPvCF;
case SSM_SS_SS_CF_C:         storage_segment_md_t  *SsSsCF;
case SSM_SS_SSRV_CF_C:       ssrv_attr_t          *SsSsrvCF;
case SSM_SS_VV_CF_C:         virtual_volume_md_t    *SsVvCF;

case SSM_ALL_ACCT_ID_C:      acct_rec_t           *AllAcctID;
case SSM_ALL_COS_ID_C:      unsigned32             *AllCosID;
case SSM_ALL_DESCNAME_C:    ssm_descname_t        *AllDescNameID;
case SSM_ALL_DEV_DRIVE_ID_C: DriveDataID_t      *AllDevDriveID;
case SSM_ALL_FILE_FAMILY_ID_C: unsigned32         *AllFileFamilyID;
case SSM_ALL_FILE_ID_C:     ssm_file_id_t         *AllFileID;
case SSM_ALL_FILESET_ID_C:  u_signed64           *AllFilesetID;
case SSM_ALL_FILESET_NAME_ID_C: ssm_fileset_name_t *AllFilesetNameID;
case SSM_ALL_FILESET_OBJ_ID_C: ns_ObjHandle_t    *AllFilesetObjID;
case SSM_ALL_HIER_ID_C:     unsigned32             *AllHierID;
case SSM_ALL_MIGR_POLICY_ID_C: unsigned32         *AllMigrPolicyID;
case SSM_ALL_PURG_POLICY_ID_C: unsigned32         *AllPurgPolicyID;
case SSM_ALL_REMOTE_SITE_ID_C: uuid_t           *AllRemoteSiteID;
case SSM_ALL_SCLASS_ID_C:   unsigned32             *AllSClassID;
case SSM_ALL_SERVER_ID_C:   uuid_t                 *AllServerID;
case SSM_BFS_BITFILE_ID_C:  hpssoid_t         *BfsBitfileID;
case SSM_BFS_BITFILE_REG_ID_C: ssm_bitfile_reg_id_t *BfsBitfileRegID;
case SSM_HDM_FILESET_ID_C:  ssm_hdm_fileset_id_t  *HdmFilesetID;
case SSM_LOGD_LOGFILE_ID_C: ssm_logfile_t       *LogdLogfileID;
case SSM_MPS_SCLASS_ID_C:   unsigned32             *MpsSClassID;
case SSM_MVR_DEVICE_ID_C:   unsigned32             *MvrDeviceID;
case SSM_PVL_DRIVE_ID_C:    drive_t              *PvlDriveID;
case SSM_PVL_REQUEST_ID_C:  job_id_t           *PvlRequestID;
case SSM_PVL_VOL_ID_C:      vol_t                  *PvlVolID;
case SSM_PVR_CART_ID_C:     cart_t                  *PvrCartID;
case SSM_SS_MAP_ID_C:       hpssoid_t              *SsMapID;
case SSM_SS_PV_ID_C:        ssm_ss_pv_t            *SsPvID;
case SSM_SS_SS_ID_C:        hpssoid_t              *SsSsID;
case SSM_SS_VV_ID_C:        hpssoid_t              *SsVvID;

case SSM_MAX_CLASS:         unsigned32             *MaxClass;

} SrvInfoUnion_t;
```

InvalidMO

Invalid managed object.

AllFileMO

Bitfile managed object and Name Server file information.

AllServerMO

Server managed object.

BfsBitfileMO

Bitfile managed object.

DmgDmgMO

DMAP Gateway managed object.

DmgFilesetMO

DMAP Gateway fileset managed object.

DmgFilesetFullMO

The full DMAP Gateway fileset managed object.

DmgFilesetListMO

List of DMAP Gateway fileset managed objects.

HdmFilesetMO

HDM fileset managed object.

LogdLogfileMO

Logfile managed object.

LsStatsMO

Location Server statistics managed object.

MmSfsMO

SFS managed object.

MpsMpsMO

MPS managed object.

MpsSClassMO

MPS storage class managed object.

MpsSClassAllMO

List of all MPS storage classes.

MvrDeviceMO

Device managed object.

MvrMvrMO

Mover managed object.

Nfs2StatsMO

NFS statistics managed object.

NsNsMO

Name Server managed object.

NsFilesetMO

Name Server fileset managed object.

NsFilesetFullMO

Full Name Server fileset managed object.

PvIDriveMO

Drive managed object.

PvIPvIMO

PVL managed object.

PvIQueueMO

Queue managed object.

PvIRequestMO

Request managed object.

PvIVoIMO

Volume managed object.

PvrCartMO

Cartridge managed object.

PvrPvrMO

PVR managed object.

SsMapMO

Map managed object.

SsPvMO

Physical volume managed object.

SsSsMO

Storage segment managed object.

SsSsrvMO

Storage Server managed object.

SsVvMO

Virtual volume managed object.

AllAcctCF

Accounting Policy Configuration File entry.

AllCosCF

Class of Service Configuration File entry.

AllDevDriveCF

Device and Drive Configuration File entries.

AllFileFamilyCF

File Family Configuration File entries.

AllHierCF

Hierarchy Configuration File entry.

AllLsCF

Location Server Policy Configuration File entry.

AIIRemoteSiteCF

Remote Site Configuration File entry.

AIMigrPolicyCF

Migration Policy Configuration File entry.

AIIPurgPolicyCF

Purge Policy Configuration File entry.

AIISClassCF

Storage Class Configuration File entry.

AIIServerCF

HPSS Server Configuration File entry.

BfsBfsC

BFS Configuration File entry.

BfsBfsegdiskCF

Bitfile Disk Segment Configuration File entry.

BfsBfsegtapeCF

Bitfile Tape Segment Configuration File entry.

BfsBitfileCF

Bitfile Configuration File entry.

DmgDmgCF

DMAP Gateway Configuration File entry.

DmgFilesetCF

DMAP Gateway Fileset Configuration File entry.

LogcLogcCF

Logging Client Configuration File entry.

LogcPolicyCF

Log Policy Configuration File entry.

LogdLogdCF

Logging Daemon Configuration File entry.

MmMmCF

Metadata Manager Monitor Configuration File entry.

Mnt1Mnt1CF

Mount Daemon Configuration File entry.

MpsMpsCF

MPS Configuration File entry.

MvrDeviceCF

Device Configuration File entry.

MvrMvrCF

Mover Configuration File entry.

Nfs2Nfs2CF

NFS Daemon Configuration File entry.

NsNsCF

Name Server Configuration File entry.

PvIDriveCF

Drive Configuration File entry.

PvIPvICF

PVL Configuration File entry.

PvIVoICF

Volume Configuration File entry.

PvrCartCF

Cartridge Configuration File entry.

PvrPvrCF

PVR Configuration File entry.

SsMapCF

Map Configuration File entry.

SsPvCF

Physical Volume Configuration File entry.

SsSsCF

Storage Segment Configuration File entry.

SsSsrvCF

Storage Server Configuration File entry.

SsVvCF

Virtual Volume Configuration File entry.

AllAcctID

Accounting Policy ID.

AllCosID

Class of service ID.

AllDescNameID

Descriptive name.

AllDevDriveID

Device/drive ID.

AllFileFamilyID

File Family ID.

AllFileID

Combination of file name and bitfile ID.

AllFilesetD

Fileset ID.

AllFilesetNameID

Fileset name.

AllFilesetObjID

Fileset object handle.

AllHierID

Hierarchy ID.

AllMigrPolicyID

Migration Policy ID.

AllPurgPolicyID

PurgePolicyID.

AllRemoteSiteID

Remote Site ID.

AllSCClassID

Storage Class ID, when reading Configuration File.

AllServerID

ServerID (UUID).

BfsBitfileID

Bitfile ID.

BfsBitfileReqID

Bitfile ID for registrations; name and hppsoid_t.

HdmFilesetID

HDM Fileset ID.

LogdLogfileID

Logfile ID.

MpsSClassID

Storage Class ID, Storage Class managed object.

MvrDeviceID

Device ID (Mover).

PvlDriveID

Drive ID (PVL).

PvlRequestID

Request ID.

PvlVolID

Volume ID.

PvrCartID

Cartridge ID.

SsMapID

Map ID.

SsPvID

Physical Volume ID.

SsSsID

Storage Segment ID.

SsVvID

Virtual Volume ID.

MaxClass

Maximum SrvInfoUnion class type, for defaults.

Clients

The System Manager and Data Server share information in many operations by means of the SrvInfoUnion_t.

8.1.3.11. Drive Data ID - DriveDataID_t

Description

Both the Mover and the PVL maintain configuration files for drives. The two files must be kept in synchronization. The Mover and PVL share a common device/drive ID for each drive, referenced by SSM as the DeviceID of the DriveDataID_t.

Format

```
typedef struct DriveDataID {
    unsigned32 DeviceID;
    unsigned32 MvrID;
    unsigned32 PvlID;
```

```
} DriveDataID_t;
```

DeviceID

Common device/drive ID of each drive.

MvrID

Index of the Mover into the Server List.

PvlID

Index of the PVL into the Server List.

Clients

The System Manager and Data Server share the DriveDataID in device and drive configuration operations.

8.3.1.12. Drive Data for Configuration Operations - DriveData_t**Description**

The Drive Data structure contains the necessary information for adding or deleting a Mover Device Configuration File and a PVL Device Configuration File entry.

Format

```
typedef struct DriveData {  
    unsigned32      MvrID;  
    unsigned32      PvlID;  
    device_desc_md_t MvrDevice;  
    drive_data_t    PvlDrive;  
} DriveData_t;
```

MvrID

Server list ID of Mover.

PvlID

Server list ID of PVL.

MvrDevice

Mover device data.

PvlDrive

PVL drive data.

Clients

The System Manager and Data Server share the Drive Data structure in device/drive configuration operations.

8.3.1.13. Cartridge Import Data - PvlImport_t**Description**

PvlCartInfo contains identifying information about a list of cartridges to be imported or exported from the PVL. PvlImport contains the parameters needed by the PVL for the import.

The MaxDrives parameter allows the user to specify the maximum number of drives to devote to the import. The System Manager implements this by creating MaxDrives threads, partitioning the list of cartridges among the threads as equally as possible, and allowing the threads to import their sublists concurrently. This, however, is no guarantee that the PVL will actually allot that many drives to the imports.

Format

```
typedef struct PvlCartInfo {
    unsigned32          NumCarts;
    [size_is(NumCarts)] cart_t  CartridgeName[*];
} PvlCartInfo_t;
```

NumCarts

Number of cartridges.

CartridgeName

List of cartridge names.

```
typedef struct PvlImport {
    unsigned32          PVRID;
    unsigned32          MaxDrives;
    media_type_t        MediaType;
    unsigned32          ImportType;
    unsigned32          Sides;
    manufacturer_t      Manu;
    lot_number_t        Lot;
} PvlImport_t;
```

PVRID

Index of PVR in the Server List.

MaxDrives

Maximum number of drives.

MediaType

Media type. Refer to the *Physical Volume Library Functions* chapter for a description of media_type_t.

ImportType

Import type.

Sides

Number of sides (partitions).

Manu

Manufacturer.

Lot

Lot.

Clients

The System Manager and Data Server exchange the Cartridge Import Data structures in operations that import and export cartridges.

8.3.1.14. Storage Server Resource Data Structure - SsResources_t**Description**

The SsPVInfo_t structure contains identifying information about a list of physical volumes to be defined in the Storage Server. The SsResources_t structure contains the parameters needed by the Storage Server to define the volumes.

Most of the parameters for the storage class creation are taken from the storage class definition. For disks, the estimated size may differ from that of the storage class definition.

Format

```
typedef struct SsPVInfo {
    unsigned32          NumPVs;
    [size_is(NumPVs)] vol_t  PVName[*];
} SsPVInfo_t;
```

NumPVs

Number of physical volumes in list.

PVName

List of physical volume names.

```
typedef struct SsResources {
    unsigned32          VVSClassId;
    acct_rec_t          Acct;
    u_signed64          EstimatedSize;
} SsResources_t;
```

VVSClassId

Storage Class ID.

Acct

Account ID.

EstimatedSize

Physical volume estimated size.

Clients

The System Manager and Data Server exchange the Resource Data structures in operations for adding and deleting resources to the Storage Server.

8.3.1.15. Storage Server Repack Structure - SsRepack_t**Description**

The SsRepack_t structure contains information about a list of virtual volumes to be repacked.

Format

```
typedef struct SsRepack {
    [string] char SourceFile[HPSS_MAX_PATH_NAME];
    [string] char TargetFile[HPSS_MAX_PATH_NAME];
    unsigned32   SClassID;
    unsigned32   NumberVVs;
    unsigned32   SSID;
```

```
    unsigned32    Flags;  
    unsigned32    Threshold;  
} SsRepack_t;
```

SourceFile

File containing list of VVs to repack.

TargetFile

File which will contain list of repacked VVs.

SClassID

Storage Class ID

NumberVVs

How many virtual volumes to reclaim.

SSID

Storage Server ID.

Flags

Repack flags.

Threshold

Target space threshold at which to stop.

The SourceFile and TargetFile parameters are not used in R3.

Clients

The System Manager and Data Server exchange the Reclaim data structures in operations for reclaiming virtual volumes.

8.3.1.16. Storage Server Reclaim Structure - SsReclaim_t

Description

The SsReclaim_t structure contains information about a list of virtual volumes to be reclaimed.

Format

```
typedef struct SsReclaim {  
    unsigned32    SSID;  
    unsigned32    PVLID;  
    unsigned32    NumberVVs;  
    unsigned32    SClassID;  
    [string] char WorkingDirectory[HPSS_MAX_PATH_NAME];  
} SsReclaim_t;
```

SSID

Storage Server ID.

PVLID

PVL ID.

NumberVVs

How many virtual volumes to reclaim.

SClassID

Storage Class ID.

WorkingDirectory

Working directory.

Clients

The System Manager and Data Server exchange the Reclaim data structures in operations for reclaiming virtual volumes.

8.3.1.17. ClientID

The ClientID is an unsigned32 which consists of two values. The BaseID, always placed in the lower 16 bits of the ClientID, represents the index of the client in the Client List kept by the System Manager. It is assigned to the client at check-in by the System Manager and required as input on all subsequent APIs. The FunctionID, placed in the upper 16 bits of the ClientID, is optionally included by the client on the ssm_AttrReg call. The R3 Data Server will use the FunctionID as a console ID, so that it can support different registrations for the same managed object for multiple consoles, but different System Manager clients are free to use the FunctionID in any way they choose. The FunctionID is never required, so clients can choose to ignore it.

The System Manager will store registrations for the same BaseID with different FunctionIDs as separate entries in the client list of the appropriate entry in SSM_SM_registered_mo. In this way, registrations and deregistrations for one console will not affect those for another console. However, if the same client has registered for more than one FunctionID for a particular managed object, that client will receive only one copy of the notification, which he is then responsible for delivering to the proper consoles as appropriate.

8.3.2. Data Private to the System Manager

8.3.2.1. Table of Registered Clients - client_list_t

Description

The table of registered clients includes the CDS name and binding handle for each client who checks in. It has a mutex and version number for protecting the connection handle; the binding handles are protected by the handles library.

Slots are available for reuse when a client checks out.

The InUse flag remains set while a client is checking out, so that no other client can manipulate the binding or connection handles. The SSM_SM_IN_USE bit of the BhState flag, however, is cleared immediately when the client begins check-out to show that no further notifications should be sent to that client.

Clients who cannot be contacted within SSM_SM_CLIENT_MAX_FAILTIME seconds are checked out automatically.

Whenever a client is successfully notified, his FailTime and Failures are cleared and his SuccessTime is reset. In order to avoid locking the mutex unnecessarily, this is only done if the FailTime has been previously set, and the check on FailTime is done before locking the mutex. Therefore, there is a window in which the failure information could be corrupted, but in practice it is not likely this would result in an automatic check-out, so the risk is taken for the sake of efficiency.

The number of queue monitor threads per client is determined by a set of environment variables specified in `hpss_env`. The set of queue monitor threads for each slot in the client table is created when a client first checks into that slot. At that point, `NotifyQInit` is set to true. If the client checks out and a different client subsequently checks into the same slot, the same queue monitor threads are used.

Format

```
typedef struct client_struct {
    unsigned32      InUse;
    char            ClientName[HPSS_MAX_DCE_NAME];
    unsigned32      NotifyQInit;
    rpc_master_handle_t Bh;
    unsigned32      BhState;
    condition_variable_t BhLock;
    hpss_connect_handle_t *Ch;
    rpc_master_handle_t ChBh;
    unsigned32      ChState;
    unsigned32      ChVersion;
    condition_variable_t ChLock;
    time_t          SuccessTime;
    time_t          FailTime;
    unsigned32      FailCount;
    pthread_mutex_t FailMutex;
} client_struct_t;
```

InUse

Whether the slot defines a valid client.

ClientName

Client's CDS name.

NotifyQInit

Whether notify queue threads and locks have been created.

Bh

Binding handle to client.

BhState

Binding state.

BhLock

Lock for Bh and BhState.

Ch

Connection handle.

ChBh

Binding handle upon which Ch is built.

ChState

State of Ch and ChBh.

ChVersion

Version number of Ch.

ChLock

Lock for Ch, ChBh, ChState, and ChVersion.

SuccessTime

Time of last success.

FailTime

Time at which connections to client began to fail.

FailCount

Count of failures since FailTime.

FailMutex

Lock for manipulating FailTime, FailCount, SuccessTime.

```
typedef struct client_list {
    client_struct_t List[SSM_MAX_CLIENT];
} client_list_t;
```

List

The list of clients.

Clients

The Client List is used internally by the System Manager.

8.3.2.2. Server Network Connection Table - server_net_t

Description

The Server Network Connection Table includes the interface specification and binding handles for each server. It has a mutex and version number for protecting the connection handle; the binding handles are protected by the handles library.

The Server Network Connection Table uses the same index for each server as the Server List. Slots are not reused.

The MonitorTh is the ID of a thread created to monitor the server's execution and connection status. The thread executes the sm_adm_check_one function. The MonitorThRC is initialized to SSM_ENO_MONITOR_TH, indicating no thread has yet been created for the server. Once the thread is created, MonitorThRC is set to SSM_SM_MONITOR_TH_RUNNING. If the server needs no monitor thread, either because it is a SNOOTY server or because its configuration has been deleted and it has been shut down, the MonitorThRC is set to 0. If the monitor thread exits for any other reason, it stores its exit value in the MonitorThRC.

Format

```
typedef union bh_primary {
    trpc_master_handle_t TRPC;
    rpc_master_handle_t RPC;
} bh_primary_t;
```

TRPC

TRPC binding handle.

RPC

RPC binding handle.

```
typedef struct server_net {
    bh_primary_t      Bh;
    hpss_connect_handle_t *Ch;
    rpc_master_handle_t ChBh;
    unsigned32        ChState;
    unsigned32        ChVersion;
    condition_variable_t ChLock;
    rpc_if_handle_t   Ifspec;
    pthread_t         MonitorTh;
    signed32          MonitorThRC;
    condition_variable_t MonitorLock;
} server_net_t;
```

Bh

Primary binding handle.

Ch

Connection handle.

ChBh

Binding handle used to build connection handle.

ChState

State of Ch and ChBh.

ChVersion

Connection handle version.

ChLock

Condition variable for Ch, ChBh, ChState, ChVersion.

Ifspec

Interface specification.

MonitorTh

Monitor thread ID.

MonitorThRC

Return code from monitor thread.

MonitorLock

Condition variable to protect MonitorTh and MonitorThRC.

```
typedef struct server_net_list {
    unsigned32      Size;
    server_net_t    List[SSM_MAX_SERVERS];
} server_net_list_t;
```

Size

Size of the list.

List

The list of servers.

Clients

The Server Network Connection Table is used internally by the System Manager.

8.3.2.3. Table of Registered Managed Object Attributes - registered_mo_t**Description**

The Table of Registered Managed Object Attributes keeps track of which clients have registered for which attributes of which managed objects for which servers.

Format

```
typedef struct registered_client {
    unsigned32      ClientID;
    u_signed64      RegisterBitmap;
    struct registered_client *Next;
} registered_client_t;
```

ClientID

Index of client in the Client List.

RegisterBitmap

Attributes of this managed object for which the client is registered.

Next

Next client in linked list of clients registered for this managed object.

```
typedef struct registered_mo {
    unsigned32      ServerID;
    unsigned32      MOClass;
    SrvInfoUnion_t *ObjectID;
    u_signed64      RegisterBitmap;
    registered_client_t *Clients;
    struct registered_mo *Next;
} registered_mo_t;
```

ServerID Index of Server in Server List.

MOClass Type of managed object.

ObjectID Object ID of the managed object.

RegisterBitmap All attributes of this managed object for which any client is registered.

Clients Linked list of clients registered for this managed object.

Next Next managed object in linked list.

Clients

The Table of Registered Managed Object Attributes is used internally by the System Manager.

8.3.2.4. Notification Queues

Description

There are five notification queues:

| | |
|------------------------------|---|
| SSM_SM_NOTIFY_Q_DATA | for managed object attribute changes. |
| SSM_SM_NOTIFY_Q_LIST | for SSM List and information notifications. |
| SSM_SM_NOTIFY_Q_LOG | for alarms, events, and status messages. |
| SSM_SM_NOTIFY_Q_TAPE | for tape mount notifications. |
| SSM_SM-NOTIFY_Q_TAPE_CHECKIN | for tape check-in notifications. |

Items are added to each queue using `sm_client_notify_q_add` by the processes which receive or package notifications:

```
sm_client_notify_info
sm_client_notify_list
ssm_LogMsgNotify
ssm_MountNotify
ssm_process_notification
ssm_TapeCheckInNotify.
```

Items are removed from each queue by one or more dedicated threads per client, each executing the `sm_client_notify_q_monitor` function. The number of threads to be created per client per queue is determined by these environment variables, defined in the `hpss_env` file:

| | |
|------------------------------------|------------------------------|
| HPSS_NOTIFY_Q_DATA_THREADS | for the data queue. |
| HPSS_NOTIFY_Q_LIST_THREADS | for the list queue. |
| HPSS_NOTIFY_Q_LOG_THREADS | for the log queue. |
| HPSS_NOTIFY_Q_TAPE_THREADS | for the tape queue. |
| HPSS_NOTIFY_Q_TAPE_CHECKIN_THREADS | for the tape check-in queue. |

The System Manager imposes reasonable limits on the number of threads.

A separate condition variable structure is defined for the notification queue, since we use it differently than the standard System Manager condition variable. In particular, we treat the Predicate as a bitmask defining which clients have just had notifications added to the queue.

We define a special copy of the log message, so that we can malloc just enough space for the actual message body and not the entire 16k allocated to a real log message, to store on our queue.

The `DataChange_t` structure referenced below is defined in the Data Structure Design Specification.

For list notifications, only a code defining the list type is stored in the queue; when the queue entry is processed, a copy of the list will be allocated then to send to the Data Server.

Each queue entry includes a bitmask of the clients to receive that notification, ClientsToNotify. Each thread which processes the queue will check the mask and process only the items for its client. It will set its bit in the ClientsInProgress mask while it attempts to send the notification, so that if other threads are processing the queue for the same client they will skip the entry. The last thread to process an entry removes it from the queue.

Format

```
typedef struct notify_queue_lock {
    pthread_mutex_t  Mutex;
    pthread_cond_t   Cond;
    u_signed64       Predicate;
} notify_queue_lock_t;
```

Mutex

Mutex to protect queue.

Cond

Condition variable to protect and signal queue.

Predicate

Mask of clients for whom a notification was added.

```
typedef struct log_msg_queue_info {
    log_rec_hdr_t  LogRecHdr;
    char           *LogRecBody;
} log_msg_queue_info_t;
```

LogRecHdr

Copy of the log record header from alarm, event, status.

LogRecBody

Log message.

```
typedef union notify_queue_info {
    log_msg_queue_info_t  LogMsg;
    DataChange_t         DataChange;
    Mount_t              Mount;
    unsigned32           Info;
    unsigned32           List;
    TapeCheckIn_t        TapeCheckIn;
} notify_queue_info_t;
```

LogMsg

Alarm, event, or status message.

DataChange

Managed object attribute notification.

Mount

Tape mount notification.

Info

Information notification.

List

List notification.

TapeCheckIn

Tape check-in notification.

```
typedef struct notify_queue {
    u_signed64      ClientsToNotify;
    u_signed64      ClientsInProgress;
    unsigned32      ServerID;
    notify_queue_info_t NotifyQInfo;
    struct notify_queue *Next;
    struct notify_queue *Prev;
} notify_queue_t;
```

ClientsToNotify

Clients to receive this notification.

ClientsInProgress

Clients which are working on this entry.

ServerID

Server who sent the notification.

NotifyQInfo

Info for building the notification.

Next

Next entry in queue.

Prev

Previous entry in queue.

Clients

The Notification Queues are used internally by the System Manager.

8.3.2.5. Configuration File List - config_file_list_t

Description

The Configuration File List lists the name of each known Encina file, the type of configuration file it is, and the server who references it.

Format

```
typedef struct config_file {
    unsigned32 InUse;
    unsigned32 CFClass;
    unsigned32 Owner;
```

```
char      ConfigFileName[HPSS_MAX_DCE_NAME];
} config_file_t;
```

InUse

Whether this slot defines a valid file.

CFClass

Type of configuration file.

Owner

Server who references the file.

ConfigFileName

File name.

```
typedef struct config_file_list {
    unsigned32      Size;
    config_file_t List[SSM_SM_MAX_CFILES];
} config_file_list_t;
```

Size

Size of the list.

List

The list of files. Refer to the definition of `server_config_list_t` provided in this chapter.

Clients

The Configuration File List is used internally by the System Manager.

8.3.2.6. Copy of the HPSS Server Configuration File - `server_config_list_t`**Description**

The `server_config_list_t` structure is a copy of the HPSS Server Configuration File. It uses the same index for each server as the Server List. Slots are not reused.

Format

```
typedef struct server_config_list {
    unsigned32      Size;
    hpss_server_config_t List[SSM_MAX_SERVERS];
} server_config_list_t;
```

Size

Size of the list.

List

The list of servers.

Clients

The HPSS Server Configuration File Copy is used internally by the System Manager.

8.3.2.7. Condition Variable Structure - `condition_variable_t`

Description

The Condition Variable Structure defines a condition variable and associated mutex and predicate.

Format

```
typedef struct condition_variable {
    pthread_mutex_t    Mutex;
    pthread_cond_t     Cond;
    unsigned32         Predicate;
} condition_variable_t;
```

Mutex

Mutex to control access to the condition variable.

Cond

Condition variable.

Predicate

Predicate.

Clients

The Condition Variable Structure is used internally by the System Manager.

8.3.2.8. Drive Data Structure - `DriveDataID_t`

Description

Configuration on PVL drives and Mover devices is done in tandem to ensure that the information on each pair is kept in sync. This structure is used as an "object ID" to identify device/drive pairs in configuration requests to the System Manager.

Format

```
typedef struct DriveDataID {
    unsigned32 DeviceID;
    unsigned32 MvrID;
    unsigned32 PvlID;
} DriveDataID_t;
```

DeviceID

Device/drive ID shared between the Mover and the PVL.

MvrID

Mover ID.

PvlID

PVL ID.

Clients

This structure is used internally by the System Manager.

8.3.2.9. Bitfile ID Register Structure - `ssm_bitfile_reg_id_t`

Description

This structure is used internally to the System Manager to associate bitfile ids (SOIDs) with bitfile names. Clients should not use this structure directly.

Format

```
typedef struct ssm_bitfile_reg_id {
    [string] char  FileName[HPSS_MAX_PATH_NAME];
    hpssoid_t     BitfileID;
} ssm_bitfile_reg_id_t;
```

FileName

The name of an HPSS bitfile.

BitfileID

The ID (SOID) of the bitfile.

8.3.2.10. Descriptive Name - ssm_descname_t**Description**

This structure is used as an "object ID" when the object needed is a server descriptive name. Currently it is used only when doing logging policy configurations.

Format

```
typedef struct ssm_descname {
    [string] char  DescName[HPSS_MAX_DESC_NAME];
} ssm_descname_t;
```

DescName

The descriptive name of an HPSS server.

Clients

This structure is used internally by the System Manager.

8.3.2.11. Bitfile Object ID - ssm_file_id_t**Description**

This structure is used as an "object ID" for operations involving bitfile managed objects. It specifies a bitfile by its name.

Format

```
typedef struct ssm_file_id {
    [string] char  FileName[HPSS_MAX_PATH_NAME];
} ssm_file_id_t;
```

FileName

The name of an HPSS bitfile.

Clients

This structure is used internally by the System Manager.

8.3.2.12. Log File Object ID - ssm_logfile_t**Description**

This structure is used as an "object ID" for operations involving logfile managed objects. It specifies the name of a logfile.

Format

```
typedef struct ssm_logfile {
    [string] char Logfile[HPSS_MAX_FILE_NAME];
} ssm_logfile_t;
```

The name of a logfile. This name does not include path information.

Clients

This structure is used internally by the System Manager.

8.3.2.13. Storage Server PV Object ID - ssm_ss_pv_t

Description

This structure is used as an "object ID" for operations involving Storage Server physical volume managed objects. It specifies the name of a physical volume.

Format

```
typedef struct ssm_ss_pv {
    [string] char PVName[PV_NAME_SIZE];
} ssm_ss_pv_t;
```

PVName

The name of a Storage Server physical volume.

Clients

This structure is used internally by the System Manager.

8.3.2.14. Site List – SiteList_t

Description

The SiteList_t structure defines an SSM Remote Site list.

Format:

The SiteList_t structure has the following format:

```
typedef struct SiteList {
    unsigned32                Size;
    unsigned32                Status;
    unsigned32                Version;
    [size_is(Size)] hpss_site_md_t List[*];
} SiteList_t;
```

Size

The number of remote sites in the list, and the size of the List conformant array.

Status

A group of flag bits describing the current status of the Remote Site list.

Version

A version number or sequence number for the list.

List

An array of HPSS Remote Site structures. `hpss_site_md_t` is an HPSS data type defined in `hpss_site.idl`.

8.3.2.15. File Family Structure – FileFamilyStruct_t**Description**

The FileFamilyStruct_t defines a single file family record.

Format

The FileFamilyStruct_t structure has the following format:

```
typedef struct FileFamilyStruct {
    unsigned32      FamilyId;
    [string] char   FamilyName[HPSS_MAX_OBJECT_NAME];
} FileFamilyStruct_t;
```

FamilyId

The ID number of the File Family.

FamilyName

The File Family name.

8.3.2.16. File Family List – FileFamilyList_t**Description**

The FileFamilyList_t structure defines an SSM File Family list.

Format

The FileFamilyList_t structure has the following format:

```
typedef struct FileFamilyList {
    unsigned32      Size;
    unsigned32      Status;
    unsigned32      Version;
    [size_is(Size)] FileFamilyStruct_t List[*];
} FileFamilyList_t;
```

Size

The number of file families in the list, and the size of the List conformant array.

Status

A group of flag bits describing the current status of the File Family list.

Version

A version number or sequence number for the list.

List

An array of SSM File Family structures. The FileFamilyStruct_t structure defines a descriptive structure for one File Family.

8.3.2.17. File Attribute Structure – ssm_fileattr_t**Description**

The `ssm_fileattr_t` structure contains the name and attributes for a file.

Format

The `ssm_fileattr_t` structure has the following format:

```
typedef struct ssm_fileattr {
    [string] char    FileName[HPSS_MAX_PATH_NAME];
    hpss_fileattr_t FileAttr;
} ssm_fileattr_t;
```

FileName

The File name.

FileAttr

The file attributes.

8.3.2.18. Logging Daemon Logfile Name Structure – `ssm_logfile_t`

Description

The `ssm_logfile_t` structure contains the name of the Logging Daemon's logfile.

Format

The `ssm_logfile_t` structure has the following format:

```
typedef struct ssm_logfile {
    [string] char    Logfile[HPSS_MAX_FILE_NAME];
} ssm_logfile_t;
```

Logfile

The logfile name.

8.3.2.19. HDM Fileset Identification Structure – `ssm_hdm_fileset_id_t`

Description

The `ssm_hdm_fileset_id_t` structure contains information about an HDM fileset.

Format

The `ssm_hdm_fileset_id_t` structure has the following format:

```
typedef struct ssm_hdm_fileset_id {
    u_signed64      FilesetID;
    unsigned32      FileSystemID;
    unsigned32      HDMPort;
    [string] char    HDMHostname[HPSS_MAX_HOST_NAME];
    [string] char    FileSystemName[HPSS_MAX_PATH_NAME];
} ssm_hdm_fileset_id_t;
```

FilesetID

The HDM fileset id.

FileSystemID

The HDM fileset file system id.

HDMPort

Port number for the HDM controlling this fileset.

HDMHostname

Host where the HDM controlling this fileset is running.

FileSystemName

Name of the file system for this fileset.

8.3.2.20. Name Server Fileset Information Structure – ssm_ns_fileset_t

Description

The ssm_ns_fileset_t structure contains information about an Name Server fileset.

Format

The ssm_ns_fileset_t structure has the following format:

```
typedef struct ssm_ns_fileset {
    ns_FilesetAttrs_t    FSAttrs;
    ns_Attrs_t           Attrs;
    ns_AttrBits_t       AttrBits;
    ns_FilesetAttrBits_t FSAttrBits;
} ssm_ns_fileset_t;
```

FSAttrs

The Name Server Fileset attributes structure.

Attrs

The Name Server attributes structure.

AttrBits

The Name Server attributes bits.

FSAttrBits

The Name Server Fileset attributes bits.

8.3.2.21. Fileset Name Structure – ssm_fileset_name_t

Description

The ssm_fileset_name_t structure contains a Fileset Name.

Format

The ssm_fileset_name_t structure has the following format:

```
typedef struct ssm_fileset_name {
    [string] char FilesetName[NS_FS_MAX_FS_NAME_LENGTH];
} ssm_fileset_name_t;
```

FilesetName

The Name Server Fileset name.

8.3.2.22. DMAP Gateway Fileset Structure – ssm_dmg_fileset_t

Description

The `ssm_dmg_fileset_t` structure contains information needed to create a DMAP Gateway Fileset .

Format

The `ssm_dmg_fileset_t` structure has the following format:

```
typedef struct ssm_dmg_fileset {
    unsigned32      CreateBoth;
    unsigned32      UID;
    unsigned32      GID;
    unsigned32      Mode;
    dmg_fileset_attr_t  FilesetAttr;
} ssm_dmg_fileset_t;
```

CreateBoth

Flag to tell the DMAP Gateway to create both the fileset in both DFS and HPSS.

UID

User id that is to own the fileset.

GID

Group id that is to own the fileset.

Mode

Default permissions for the fileset.

FilesetAttr

Attributes that are needed to create the fileset.

8.4. Data Server Client Interfaces

The interface below is for any Data Server clients of the System Manager

Data Server Clients are expected to provide the System Manager a **client_Notify** API to receive asynchronous notifications. Clients who do not provide this api will be automatically checked out by the System Manager.

8.4.1. client_Notify

Purpose

DCE server function for SSM Data Server.

Syntax

```
#include "ssm_ds_if.h"

signed32 client_Notify(
    handle_t                bh,           /* IN */
    hpss_connect_handle_t  *ch,         /* IN */
    unsigned32              server_id,   /* IN */
    NotifyUnion_t          *notification, /* IN */
    error_status_t         *RPCstatus    /* OUT */
);
```

Description

This function is called by the SSM System Manager via DCE RPC's in order to asynchronously notify the Data Server of:

- log messages from the logging daemon
- data changes from HPSS servers
- mount and tape check-in requests from PVRs
- new lists (e.g., the server list) from the System Manager
- miscellaneous requests from the System Manager

Because Sammi is not thread-safe, all Sammi functions have been isolated to one thread. Therefore, this function does nothing except receive notification packets from the System Manager, reformat them into more convenient form, and place them into a singly-linked FIFO list for later processing by the Sammi thread.

Parameters

| | |
|---------------------|--|
| <i>bh</i> | Explicit binding handle. |
| <i>ch</i> | Pointer to an HPSS connection handle. |
| <i>server_id</i> | Server array index of server issuing the notification. |
| <i>notification</i> | Pointer to a notification union structure. |
| <i>RPCstatus</i> | DCE RPC status return. |

Return values

Upon successful completion, a value of zero (0) is returned. If an error occurs, a value of -1 is returned.

Error conditions

The only error condition returned is -1 to indicate that the caller passed invalid information in the server ID or notification arguments, or that the caller is not authorized to call the API.

See also

None.

Clients

System Manager

Notes

None.

8.5. Other Data Definitions (Data Server Clients)

This section describes key data definitions that are used by Data Server clients.

A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

8.5.1. Data Server Notification structure - `NotifyUnion_t`

Description

The `NotifyUnion_t` structure is used to pass information from the SSM System Manager to the SSM Data Server and to other clients of the System Manager. It is an argument for the `client_Notify` API.

Format

The `NotifyUnion_t` structure has the following format:

```
typedef union switch (unsigned32 NotifyClass) NotifyData {
    case SSM_ALARM_N:          log_msg_t      Alarm;
    case SSM_EVENT_N:         log_msg_t      Event;
    case SSM_STATUS_N:        log_msg_t      Status;
    case SSM_DATA_CHANGE_N:   DataChange_t   DataChange;
    case SSM_MOUNT_N:         Mount_t        Mount;
    case SSM_INFO_N:          unsigned32     Info;
    case SSM_LIST_N:          ListUnion_t    List;
    case SSM_TAPE_CHECKIN_N:  TapeCheckIn_t  TapeCheckIn;
} NotifyUnion_t;
```

Alarm

The `log_msg_t` structure is defined by the Logging Services.

Event

The `log_msg_t` structure is defined by the Logging Services.

Status

The `log_msg_t` structure is defined by the Logging Services.

Data Change

The `DataChange_t` structure is used to send a data change notification (a change in a server attribute for which notifications have been requested by setting a bit in the server's registration bitmap).

Format

The `DataChange_t` structure has the following format:

```
typedef struct DataChange {
    unsigned64      RegisterBitmap;
    SrvInfoUnion_t  Attribute;
} DataChange_t;
```

RegisterBitmap

A 64-bit registration bitmap indicating which managed attributes have changed.

Attribute

The managed object sent from the notifying server, with the attributes marked by `RegisterBitmap` filled in with the changed data. The `SrvInfoUnion_t` structure is defined in this chapter.

Mount

The Mount_t structure is used to send mount request notifications.

Format

The Mount_t structure has the following format:

```
typedef struct Mount {
    unsigned32      PVRID;
    cart_t          *Cart;
    side_t          *Side;
    drive_addr_t    *Drive;
    unsigned32      RequestType;
} Mount_t;
```

PVRID

Index into the SSM server array of the PVR issuing the mount request.

Cart

A pointer to a cartridge ID structure. The cart_t data type is defined in the PVR chapter.

Side

A pointer to a cartridge side specifier. The side_t data type is defined in the PVR chapter.

Drive

A pointer to a drive address. The drive_addr_t data type is defined in the PVR chapter.

RequestType

A constant defining the request type.

Info

The Info type of notification is used by the SSM System Manager to notify the Data Server (or other client) of some event of interest only between the System Manager and its clients. The value of Info is set to a constant describing the type of event that has occurred.

List

The ListUnion_t structure is used to send new metadata list structures, such as the server list or device/drive list.

Format

The ListUnion_t structure has the following format:

```
typedef union switch (unsigned32 ListClass) ListData {
    case SSM_LIST_SERVER:      ServerList_t      *ServerList;
    case SSM_LIST_COS:        COSList_t         *COSList;
    case SSM_LIST_DRIVE:      DriveList_t        *DriveList;
    case SSM_LIST_SCLASS:     SClassList_t       *SClassList;
    case SSM_LIST_HIER:       HierList_t         *HierList;
    case SSM_LIST_MIGRP:      MigrPList_t        *MigrPList;
    case SSM_LIST_PURGP:      PurgPList_t        *PurgPList;
    case SSM_LIST_SITE:       SiteList_t         *SiteList;
    case SSM_LIST_FILE_FAMILY: FileFamilyList_t  *FileFamilyList;
} ListUnion_t;
```

ServerList

A pointer to a server list. The ServerList_t structure defines an SSM server list. Refer to the

Server List - *ServerList_t* section in this chapter for the format of this structure.

COSList

A pointer to a Class Of Service list. The *COSList_t* structure defines an SSM Class Of Service list. Refer to the *Class of Service List - COSList_t* section in this chapter for the format of this structure.

DriveList

A pointer to a Class Of Service list. The *DriveList_t* structure defines an SSM Class Of Service list. Refer to the *Drive List - DriveList_t* section in this chapter for the format of this structure.

SClassList

A pointer to a Storage Class list. The *SClassList_t* structure defines an SSM Storage Class list. Refer to the *Storage Class List - SClassList_t* section in this chapter for the format of this structure.

HierList

A pointer to a Storage Hierarchy list. The *HierList_t* structure defines an SSM Storage Hierarchy list. Refer to the *Hierarchy List - HierList_t* section in this chapter for the format of this structure.

MigrPList

A pointer to a Migration Policy list. The *MigrPList_t* structure defines an SSM Migration Policy list. Refer to the *Migration Policy List - MigrPList_t* section in this chapter for the format of this structure.

PurgPList

A pointer to a Purge Policy list. The *PurgPList_t* structure defines an SSM Purge Policy list. Refer to the *Migration Policy List - PurgPList_t* section in this chapter for the format of this structure.

SiteList:

A pointer to a Remote HPSS Site list. The *SiteList_t* structure defines an SSM Remote Site list.

FileFamilyList:

A pointer to a File Family list. The *FileFamilyList_t* structure defines an SSM File Family list.

TapeCheckIn

The *TapeCheckIn_t* structure is used to send tape check-in request notifications.

Format

The *TapeCheckIn_t* structure has the following format:

```
typedef struct TapeCheckIn {
    unsigned32      PVRID;
    cart_t          *Cart;
    side_t          *Side;
    ioport_addr_t  *IOPort;
    unsigned32      RequestType;
} TapeCheckIn_t;
```

PVRID

Index into the SSM server array of the PVR issuing the tape check-in request.

Cart

A pointer to a cartridge ID structure. The `cart_t` data type is defined in the PVR chapter.

Side

A pointer to a cartridge side specifier. The `side_t` data type is defined in the PVR chapter.

IOPort

A pointer to a ioport address. The `ioport_addr_t` data type is defined in the PVR chapter.

RequestType

A constant defining the request type.

9. Location Server Functions

This chapter specifies the Location Server programming interface. Specifically, the following information is provided:

Client Cache Programming Interface (CCPIs)

Server Programming Interface (SPIs)

Data Definitions

9.1. Client Cache Programming Interface Functions

This section describes all APIs which are provided for access through a client side cache for use by another HPSS subsystem or by a client external to HPSS. The CCPI interface specification includes the following information:

Name

Syntax

Description

Parameters

Return Values

Error Conditions

Related Information

Clients

Notes

9.1.1. hpss_LocateBFSByCOSHints**Purpose**

Obtain COS and BFS information given COS hints through a local cache.

Syntax

```
#include "hpss_ls.h"
```

```
signed32
```

```
hpss_LocateBFSByCOSHints(
```

```
    signed32      RequestID,    /* IN */
    uuid_t        HPSSId,      /* IN */
    hpss_cos_hints_t *COSHints, /* IN */
    hpss_cos_priorities_t *COSPrio, /* IN */
    unsigned32    SortMethod,   /* IN */
    unsigned32    MaxReturned, /* IN */
    ls_cos_bfs_array_t **COSBFS) /* OUT */
```

Description

Given COS hints, the `hpss_LocateBFSByCOSHints` function returns a weighted list of COS/BFS pairs which match. This routine is called by the client API during file creation to determine which COS and BFS to use. If the results are in the local cache, the results are returned immediately. Otherwise the local Location Server is contacted and the results are placed in the local cache before being returned. By default, the COS/BFS pairs (in each element returned) are sorted by the `LS_SORT_BY_WEIGHT` method described below.

Parameters

| | |
|--------------------|--|
| <i>RequestID</i> | Request ID for the current request. |
| <i>HPSSId</i> | HPSS ID of the HPSS system to contact. A nil UUID value defaults to the local HPSS system. |
| <i>COSHints</i> | Pointer to a COS Hints structure. If this is NULL, denoting the default COS, the <i>COSPrio</i> field must also be NULL. |
| <i>COSPrio</i> | Pointer to a COS Priorities structure. If this is NULL, the <i>COSHints</i> field must also be NULL. |
| <i>SortMethod</i> | Method used to sort the returned elements. A value of zero (0) sorts the returned elements according to the default sort method. Other valid values are: LS_SORT_BY_COS: Sort by COS then BFS. LS_SORT_BY_WEIGHT: Sort by Weight, COS then BFS. This is the default sort method. |
| <i>MaxReturned</i> | Specifies the maximum number of elements to return in COSBFS. If this is zero (0), all elements that match will be returned. |
| <i>COSBFS</i> | Array of COS/BFS pairs which match the hints passed in. |

Return values

Upon successful completion, `hpss_LocateBFSByCOSHints` returns zero. Otherwise, it returns one of the following error conditions described below.

Error conditions

| | |
|-------------------------------|--|
| <code>HPSS_ECONN</code> | Connection problem contacting the Location Server. |
| <code>HPSS_EFAULT</code> | <code>COSBFS</code> is NULL. |
| <code>HPSS_EINVAL</code> | Invalid parameter passed in. |
| <code>HPSS_EINVALCOS</code> | Invalid COS information specified. |
| <code>HPSS_EINVALHINTS</code> | Invalid Hints information specified. |
| <code>HPSS_ENOENT</code> | No COS or BFS was found that matched hints passed in. |
| <code>HPSS_ENOMEM</code> | Not enough memory exists to initialize the client cache. |

See also

`Is_BFSByCOSHints`.

Clients

Client API.

Notes

If the `HPSSId` parameter specifies a remote HPSS system, a Location Server at that site is located and contacted by the local Location Server to obtain the COS/BFS information.

The `Weight` fields returned for the elements in `COSBFS` are only meaningful when compared with each other, with the highest value denoting the best match.

9.1.2. `hpss_LocateLocationServer`

Purpose

Obtain the RPC group name for contacting Location Servers at a site.

Syntax

```
#include "hpss_Is.h"
```

```
signed32
```

```
hpss_LocateLocationServer(  
    signed32
```

```
    RequestID, /* IN */
```

```
    uuid_t     HPSSId, /* IN */
```

```
    Is_map_t   *Location) /* OUT */
```

Description

The `hpss_LocateLocationServer` function returns a local or remote DCE RPC group name for Location Servers at a site, given the HPSS ID of the HPSS system. If the entry is found in the local cache, it is returned. If it is not found, the local Location Server is contacted, and the result is placed into the cache and then returned.

Parameters

RequestID Request ID for the current request.

HPSSId HPSS ID of the HPSS system to contact. A nil UUID value defaults to the local HPSS system.

Location Location information for the DCE RPC Group.

Return values

Upon successful completion, `hpss_LocateLocationServer` returns zero. Otherwise, it returns one of the following error conditions described below.

Error conditions

HPSS_ECONN Connection problem contacting the Location Server.

HPSS_EFAULT Location is NULL.

HPSS_ENOENT No information exists for the location specified by HPSSId.

HPSS_ENOMEM Not enough memory exists to initialize the client cache.

See also

`Is_LocationServer`.

Clients

None.

Notes

None.

9.1.3. hpss_LocateRootNS**Purpose**

Obtain the local root Name Server's location information through a local cache.

Syntax

```
#include "hpss_Is.h"
```

```
signed32
```

```
hpss_LocateRootNS(
```

```
    signed32
```

```
    RequestID,    /* IN */
```

```
    Is_map_t
```

```
    *Location)   /* OUT */
```

Description

The `hpss_LocateRootNS` returns the local root Name Server's location map information. If the entry is found in the local cache, it is returned. If it is not found, the Location Server is contacted, and the result is placed into the cache and then returned.

Parameters

RequestID Request ID for the current request.

Location Location information for the local HPSS root Name Server.

Return values

Upon successful completion, `hpss_LocateRootNS` returns zero. Otherwise, it returns one of the following error conditions described below.

Error conditions

HPSS_ECONN Connection problem contacting the Location Server.

HPSS_EFAULT Location is NULL.

HPSS_ENOENT No local root Name Server has been defined.

HPSS_ENOMEM Not enough memory exists to initialize the client cache.

See also

`Is_RootNS`.

Clients

Client API.

Notes

None.

9.1.4. `hpss_LocateServerByPath`

Purpose

Obtain an HPSS server's location information from its CDS Path through a local cache.

Syntax

```
#include "hpss_Is.h"
```

```
signed32
```

```
hpss_LocateServerByPath(
```

```
    signed32 RequestID, /* IN */  
    char *CDSPath, /* IN */  
    Is_map_t *Location) /* OUT */
```

Description

The `hpss_LocateServerByPath` function maps an HPSS Server's CDS path to its location information. If the entry is found in the local cache, it is returned. If it is not found, the Location Server is contacted, and the result is placed into the cache and then returned.

Parameters

| | |
|------------------|-------------------------------------|
| <i>RequestID</i> | Request ID for the current request. |
| <i>CDSPath</i> | CDS Pathname of the HPSS Server. |
| <i>Location</i> | HPSS Server location information. |

Return values

Upon successful completion, `hpss_LocateLocationServer` returns zero. Otherwise, it returns one of the following error conditions described below.

Error conditions

| | |
|-------------|--|
| HPSS_ECONN | Connection problem contacting the Location Server. |
| HPSS_EFAULT | CDSPath or Location is NULL. |
| HPSS_EINVAL | The CDSPath string is empty or not a valid CDS path. |
| HPSS_ENOENT | No location information exists for that CDS path. |
| HPSS_ENOMEM | Not enough memory exists to initialize the client cache. |

See also

`hpss_LocateServerByUUID`, `Is_ServerByPath`.

Clients

None.

Notes

None.

9.1.5. hpss_LocateServerByUUID**Purpose**

Obtain an HPSS server's location information from its UUID through a local cache.

Syntax

```
#include "hpss_Is.h"
```

```
signed32
```

```
hpss_LocateServerByUUID(
```

```
    signed32 RequestID, /* IN */
    uuid_t   UUID,     /* IN */
    Is_map_t *Location) /* OUT */
```

Description

The `hpss_LocateServerByUUID` function looks up an HPSS server's location by its UUID. If the entry is found in the local cache, it is returned. If it is not found, the Location Server is contacted, and the result is placed into the cache and then returned.

Parameters

| | |
|------------------|--|
| <i>RequestID</i> | Request ID for the current request. |
| <i>UUID</i> | Universal ID of HPSS Server to locate. |
| <i>Location</i> | HPSS Server location information. |

Return values

Upon successful completion, `hpss_LocateServerByUUID` returns zero. Otherwise, it returns one of the following error conditions described below.

Error conditions

| | |
|-------------|--|
| HPSS_ECONN | Connection problem contacting the Location Server. |
| HPSS_EFAULT | Location is NULL. |
| HPSS_EINVAL | The UUID is invalid or is zeroed out (nil). |
| HPSS_ENOENT | No information exists for that UUID. |
| HPSS_ENOMEM | Not enough memory exists to initialize the client cache. |

See also

`hpss_LocateServerByPath`, `Is_ServerByUUID`.

Clients

Client API.

Notes

None.

9.1.6. `hpss_LocationLibInit`

Purpose

Initialize the Location Client Cache Library.

Syntax

```
#include "hpss_Is.h"
```

```
signed32  
hpss_LocateServerInit(void)
```

Description

The `hpss_LocationLibInit` function initializes the Location Client Cache Library for use. Calling this function is optional since each of the Location Client Cache Library routines call this function internally if initialization is needed.

Parameters

None

Return values

Upon successful completion, `hpss_LocationLibInit` returns zero. Otherwise, it returns one of the following error conditions described below.

Error conditions

| | |
|--------------------------|--|
| <code>HPSS_ECONN</code> | Connection problem contacting the Location Server. |
| <code>HPSS_ENOMEM</code> | Not enough memory exists to initialize the client cache. |

See also

`hpss_LocationLibDeinit`.

Clients

Client Cache Library.

Notes

None.

9.1.7. hpss_LocationLibDeinit**Purpose**

Uninitialize the Location Client Cache Library.

Syntax

```
#include "hpss_Is.h"

signed32
hpss_LocateServerDeinit(void)
```

Description

The `hpss_LocationLibDeinit` function uninitializes the Location Client Cache Library. After this function is called, no further calls should be made to the Location Client Cache Library.

Parameters

None

Return values

Upon successful completion, `hpss_LocationLibDeinit` returns zero. Otherwise, it returns one of the following error conditions described below.

Error conditions

| | |
|-------------|--|
| HPSS_ECONN | Connection problem contacting the Location Server. |
| HPSS_EINVAL | The Client Cache Library is not currently initialized. |

See also

`hpss_LocationLibInit`, `hpss_LocationLibSetConfig`.

Clients

None.

Notes

The only reason that this routine should be called is if the caller desires to close down any connection to the local Location Server. Unless specifically requested, with the `hpss_LocationLibSetConfig` function, the Client Cache Library does not maintain open connections to the Location Server so there is usually no reason to do this.

9.1.8. `hpss_LocationLibGetConfig`

Purpose

Retrieve the current configuration of the Client Cache Library.

Syntax

```
#include "hpss_Is.h"
```

```
signed32
```

```
hpss_LocationLibGetConfig(  
    Is_lib_config_t *Config)    /* OUT */
```

Description

The `hpss_LocationLibGetConfig` function returns the current Client Cache Library configuration. This is useful when called before the Client Cache Library is initialized so that the caller can alter the behavior of the library. See `hpss_LocationLibSetConfig` for more details.

Parameters

Config Configuration block returned.

Return values

Upon successful completion, `hpss_LocationLibGetConfig` returns zero. Otherwise, it returns one of the following error conditions described below.

Error conditions

HPSS_EFAULT Config is NULL.

HPSS_EINVAL The environment variable denoting the Location Server group has been set up improperly.

See also

`hpss_LocationLibSetConfig`.

Clients

None.

Notes

None.

9.1.9. hpss_LocationLibSetConfig**Purpose**

Set the current configuration of the Client Cache Library.

Syntax

```
#include "hpss_ls.h"

signed32
hpss_LocationLibSetConfig(
    ls_lib_config_t *Config) /* IN */
```

Description

The `hpss_LocationLibSetConfig` function sets the current Client Cache Library configuration. This is useful when called before the Client Cache Library is initialized so that the caller can alter the behavior of the library. Before calling this routine the `hpss_LocationLibGetConfig` function should be called to retrieve the default configuration.

Parameters

Config Configuration to set.

Return values

Upon successful completion, `hpss_LocationLibSetConfig` returns zero. Otherwise, it returns one of the following error conditions described below.

Error conditions

| | |
|-------------|---|
| HPSS_EFAULT | Config is NULL. |
| HPSS_EINVAL | The library has already been initialized and the configuration cannot be changed, or one or more of the fields of the Config parameter are invalid. |

See also

`hpss_LocationLibGetConfig`.

Clients

None.

Notes

None.

9.2. Server Programming Interface Functions

This section describes all APIs which are provided for direct access to the Location Server for use by another HPSS subsystem or by a client external to HPSS. The SPI interface specification includes the following information:

Name

Syntax

Description

Parameters

Return Values

Error Conditions

Related Information

Clients

Notes

Common Error Conditions

The following error conditions may be returned by any SPI in addition to the ones described under the individual SPIs.

| | |
|----------------|--|
| HPSS_EAGAIN | A resource is busy. The request should be retried after a delay. |
| HPSS_EBADCONN | The connection handle is bad. Reconnect and retry the request. |
| HPSS_EBUSY | The Location Server is under heavy load. Rebind to a replicated Location Server or wait a short period of time before retrying the request. |
| HPSS_ENOTREADY | The Location Server is starting up or reinitializing and is not ready to process requests. Retry the request after a short delay. |
| HPSS_EPERM | The caller does not have sufficient permissions to perform the operation. The caller must have control permission on the Location Server's Security ACL to perform administrative functions such as <code>ls_ServerSetAttrs</code> . All non-administrative functions require read permission. |
| HPSS_ESYSTEM | An internal Location Server error has occurred. If possible, rebind to a replicated Location Server and retry the request. |

9.2.1. Is_BFSByCOSHints

Purpose

Obtain COS and BFS information given COS hints.

Syntax

```
#include "ls_interface.h"
```

```
signed32
```

```
ls_BFSByCOSHints(
    handle_t                Binding,           /* IN */
    hpss_connect_handle_t  *Connect,         /* IN */
    signed32                RequestID,        /* IN */
    uuid_t                  HPSSId,          /* IN */
    hpss_cos_hints_t        *COSHints,       /* IN */
    hpss_cos_priorities_t  *COSPrio,        /* IN */
    unsigned32              SortMethod,      /* IN */
    unsigned32              MaxReturned,     /* IN */
    ls_cos_bfs_array_t      **COSBFS,       /* OUT */
    u_signed64              *MaxFileSizeHint, /* OUT */
    ls_cos_bfs_array_t      *MinFileSizeHint /* OUT */
)
```

Description

Given COS hints, the `ls_BFSByCOSHints` function returns a weighted list of COS/BFS pairs which match. This routine is normally called during file creation to determine which COS and BFS to use. By default, the COS/BFS pairs (in each element returned) are sorted by the `LS_SORT_BY_WEIGHT` method described below.

Parameters

| | |
|-------------------|--|
| <i>Binding</i> | RPC binding handle to Location Server to contact. |
| <i>Connect</i> | Optional connection handle. |
| <i>RequestID</i> | Request ID for the current request. |
| <i>HPSSId</i> | HPSS ID of the HPSS system to contact. A nil UUID value defaults to the local HPSS system. |
| <i>COSHints</i> | Pointer to a COS Hints structure. If this is NULL, denoting the default COS, the <i>COSPrio</i> field must also be NULL. |
| <i>COSPrio</i> | Pointer to a COS Priorities structure. If this is NULL, the <i>COSHints</i> field must also be NULL. |
| <i>SortMethod</i> | Method used to sort the returned elements. A value of zero (0) sorts the returned elements according to the default sort method. Other valid values are: <code>LS_SORT_BY_COS</code> : Sort by COS then BFS. <code>LS_SORT_BY_WEIGHT</code> : Sort by Weight, COS then BFS. This is the default sort method. |

| | |
|------------------------|--|
| <i>MaxReturned</i> | Specifies the maximum number of elements to return in COSBFS. If this is zero (0), all elements that match will be returned. |
| <i>COSBFS</i> | Array of COS/BFS pairs which match the hints passed in. |
| <i>MinFileSizeHint</i> | The minimum file size that the COSBFS information is valid for. Used as a hint by the Client Cache library. |
| <i>MaxFileSizeHint</i> | The maximum file size that the COSBFS information is valid for. Used as a hint by the Client Cache library. |

Return values

Upon successful completion, `Is_BFSByCOSHints` returns zero. Otherwise, it returns a common error condition or one of the following error conditions described below.

Error conditions

| | |
|------------------|---|
| HPSS_EFAULT | COSBFS, <code>MinFileSizeHint</code> or <code>MaxFileSizeHint</code> is NULL. |
| HPSS_EINVAL | Invalid <code>HPSSId</code> or <code>SortMethod</code> . |
| HPSS_EINVALCOS | Invalid COS information specified. |
| HPSS_EINVALHINTS | Invalid Hints information specified. |
| HPSS_ENOENT | No COS or BFS was found that matched hints passed in. |
| HPSS_ENOMEM | Not enough memory exists to return COSBFS array. |

See also

`hpss_LocateBFSByCOSHints`.

Clients

Client Cache Library.

Notes

Normally the `hpss_LocateBFSByCOSHints` routine should be called instead since it performs the same function as this routine, with automatic retry and rebind logic, through a client side cache to reduce network traffic.

If the `HPSSId` parameter specifies a remote HPSS system, a Location Server at that site is located and contacted by the local Location Server to obtain the COS/BFS information.

The `Weight` fields returned for the elements in `COSBFS` are only meaningful when compared with each other, with the highest value denoting the best match.

9.2.2. Is_GetServerMaps**Purpose**

Retrieve Location Server map information from a remote Location Server.

Syntax

```
#include "Is_peerIF.h"

signed32
Is_GetServerMaps(
    handle_t                Binding,        /* IN */
    hpss_connect_handle_t  *Connect,      /* IN */
    signed32                RequestID,     /* IN */
    unsigned32              Flags,        /* IN */
    Is_map_array_ptr_t     *Locations)    /* OUT */
```

Description

The Is_GetServerMaps function is used by a local Location Server to query a remote Location Server for its location map information. It does not normally need to be called outside of the Location Server itself.

Parameters

| | |
|------------------|--|
| <i>Binding</i> | RPC binding handle to Location Server to contact. |
| <i>Connect</i> | Optional connection handle. |
| <i>RequestID</i> | Request ID for the current request. |
| <i>Flags</i> | One or more of the following values OR'd together: LS_GETMAPS_LOCAL: Return maps local to remote Location Server's site. LS_GETMAPS_REMOTE: Return maps foreign to the remote Location Server's site. This is normally not used. |
| <i>Locations</i> | Returned conformant array of location maps. |

Return values

Upon successful completion, Is_GetServerMaps returns zero. Otherwise, it returns a common error condition or one of the following error conditions described below.

Error conditions

| | |
|-------------|---|
| HPSS_EFAULT | Locations is NULL. |
| HPSS_EINVAL | An invalid parameter has been passed. |
| HPSS_ENOENT | No information exists at the remote site for the Flags specified. |

See also

None.

Clients

Location Server.

Notes

None.

9.2.3. Is_LocationServer**Purpose**

Obtain the RPC group name for contacting Location Servers at a site.

Syntax

```
#include "Is_interface.h"
```

```
signed32
```

```
Is_LocationServer(
    handle_t           Binding,           /* IN */
    hpss_connect_handle_t *Connect,      /* IN */
    signed32           RequestID,        /* IN */
    uuid_t             HPSSId,           /* IN */
    Is_map_t           *Location)        /* OUT */
```

Description

The `hpss_LocateLocationServer` function returns a local or remote DCE RPC group name for Location Servers at a site, given the HPSS ID of the HPSS system.

Parameters

| | |
|------------------|--|
| <i>Binding</i> | RPC binding handle to Location Server to contact. |
| <i>Connect</i> | Optional connection handle. |
| <i>RequestID</i> | Request ID for the current request. |
| <i>HPSSId</i> | HPSS ID of the HPSS system to contact. A nil UUID value defaults to the local HPSS system. |
| <i>Location</i> | Location information for the DCE RPC Group. |

Return values

Upon successful completion, `Is_LocationServer` returns zero. Otherwise, it returns a common error condition or one of the following error conditions described below.

Error conditions

| | |
|-------------|---|
| HPSS_EFAULT | Location is NULL. |
| HPSS_ENOENT | No information exists for the site specified by HPSSId. |

See also

`hpss_LocateLocationServer`.

Clients

Client Cache Library.

Notes

Normally the `hpss_LocateLocateServer` routine should be called instead since it performs the same function as this routine, with automatic retry and rebind logic, through a client side cache to reduce network traffic.

9.2.4. Is_RootNS

Purpose

Obtain the local root Name Server's location.

Syntax

```
#include "Is_interface.h"

signed32
Is_RootNS(
    handle_t          Binding,          /* IN */
    hpss_connect_handle_t *Connect,    /* IN */
    signed32          RequestID,       /* IN */
    Is_map_t          *Location)       /* OUT */
```

Description

The Is_RootNS returns the local root Name Server's location map information

Parameters

| | |
|------------------|---|
| <i>Binding</i> | RPC binding handle to Location Server to contact. |
| <i>Connect</i> | Optional connection handle. |
| <i>RequestID</i> | Request ID for the current request. |
| <i>Location</i> | Location information for the local HPSS root Name Server. |

Return values

Upon successful completion, Is_RootNS returns zero. Otherwise, it returns a common error condition or one of the following error conditions described below.

Error conditions

| | |
|-------------|---|
| HPSS_EFAULT | Location is NULL. |
| HPSS_ENOENT | No local root Name Server has been defined. |

See also

hpss_LocateRootNS.

Clients

Client Cache Library.

Notes

Normally the hpss_LocateRootNS routine should be called instead since it performs the same function as this routine, with automatic retry and rebind logic, through a client side cache to reduce network traffic.

9.2.5. Is_ServerByPath

Purpose

Obtain an HPSS server's location information from its CDS.

Syntax

```
#include "Is_interface.h"

signed32
Is_ServerByPath(
    handle_t          Binding,          /* IN */
    hpss_connect_handle_t *Connect,    /* IN */
    signed32          RequestID,       /* IN */
    char              *CDSPath,        /* IN */
    Is_map_t          *Location)       /* OUT */
```

Description

The Is_ServerByPath function maps an HPSS Server's CDS path to its location information.

Parameters

| | |
|------------------|---|
| <i>Binding</i> | RPC binding handle to Location Server to contact. |
| <i>Connect</i> | Optional connection handle. |
| <i>RequestID</i> | Request ID for the current request. |
| <i>CDSPath</i> | CDS Pathname of the HPSS Server. |
| <i>Location</i> | HPSS Server location information. |

Return values

Upon successful completion, Is_LocationServer returns zero. Otherwise, it returns a common error condition or one of the following error conditions described below.

Error conditions

| | |
|-------------|--|
| HPSS_EFAULT | CDSPath or Location is NULL. |
| HPSS_EINVAL | The CDSPath string is empty or not a valid CDS path. |
| HPSS_ENOENT | No location information exists for that CDS path. |

See also

hpss_LocateServerByPath.

Clients

Client Cache Library.

Notes

Normally the hpss_LocateServerByPath routine should be called instead since it performs the same function as this routine, with automatic retry and rebind logic, through a client side cache to reduce network traffic.

9.2.6. Is_ServerByUUID

Purpose

Obtain an HPSS server's location information from its UUID.

Syntax

```
#include "Is_interface.h"

signed32
Is_ServerByUUID(
    handle_t                Binding,        /* IN */
    hpss_connect_handle_t *Connect,       /* IN */
    signed32                RequestID,     /* IN */
    uuid_t                  UUID,         /* IN */
    Is_map_t                *Location)    /* OUT */
```

Description

The Is_ServerByUUID function looks up an HPSS server's location by its UUID.

Parameters

| | |
|------------------|---|
| <i>Binding</i> | RPC binding handle to Location Server to contact. |
| <i>Connect</i> | Optional connection handle. |
| <i>RequestID</i> | Request ID for the current request. |
| <i>UUID</i> | Universal ID of HPSS Server to locate. |
| <i>Location</i> | HPSS Server location information. |

Return values

Upon successful completion, Is_ServerByUUID returns zero. Otherwise, it returns a common error condition or one of the following error conditions described below.

Error conditions

| | |
|-------------|---|
| HPSS_EFAULT | Location is NULL. |
| HPSS_EINVAL | The UUID is invalid or is zeroed out (nil). |
| HPSS_ENOENT | No information exists for that UUID. |

See also

hpss_LocateServerByUUID.

Clients

Client Cache Library.

Notes

Normally the hpss_LocateServerByUUID routine should be called instead since it performs the same function as this routine, with automatic retry and rebind logic, through a client side cache to reduce network traffic.

9.2.7. Is_ServerGetAttrs

Purpose

Retrieve Location Server attributes.

Syntax

```
#include "Is_interface.h"
```

```
signed32
```

```
Is_ServerGetAttrs(  
    handle_t                Binding,        /* IN */  
    hpss_connect_handle_t  *Connect,      /* IN */  
    hpss_server_attrib_t   *ServerData)   /* OUT */
```

Description

The Is_ServerGetAttrs function retrieves the server attributes of the Location Server. The caller must have control permissions on the Location Server's Security ACL in order to call this function.

Parameters

| | |
|-------------------|---|
| <i>Binding</i> | RPC binding handle to Location Server to contact. |
| <i>Connect</i> | Optional connection handle. |
| <i>ServerData</i> | Returned server attributes. |

Return values

Upon successful completion, Is_ServerGetAttrs returns zero. Otherwise, it returns a common error condition or one of the following error conditions described below.

Error conditions

| | |
|-------------|---------------------|
| HPSS_EFAULT | ServerData is NULL. |
|-------------|---------------------|

See also

Is_ServerSetAttrs.

Clients

SSM.

Notes

None.

9.2.8. Is_ServerSetAttrs

Purpose

Set Location Server attributes.

Syntax

```
#include "Is_interface.h"
```

```
signed32
```

```
Is_ServerSetAttrs(  
    handle_t                Binding,           /* IN */  
    hpss_connect_handle_t  *Connect,         /* IN */  
    u_signed64              InSelectBitmap,   /* IN */  
    u_signed64              *OutSelectBitmap, /* OUT */  
    hpss_server_attr_t     *InAttributes,    /* IN */  
    hpss_server_attr_t     *OutAttributes)   /* OUT */
```

Description

The `Is_ServerSetAttrs` function sets the server attributes of the Location Server. The caller must have control permissions on the Location Server's Security ACL in order to call this function.

Parameters

| | |
|------------------------|---|
| <i>Binding</i> | RPC binding handle to Location Server to contact. |
| <i>Connect</i> | Optional connection handle. |
| <i>InSelectBitmap</i> | Bitmap of attributes to set. |
| <i>OutSelectBitmap</i> | Returned bitmap of attributes actually set. |
| <i>InAttributes</i> | Attributes to set. |
| <i>OutAttributes</i> | Returned attributes actually set. |

Return values

Upon successful completion, `Is_ServerSetAttrs` returns zero. Otherwise, it returns a common error condition or one of the following error conditions described below.

Error conditions

| | |
|-------------|---|
| HPSS_EFAULT | <code>OutSelectBitmap</code> , <code>InAttributes</code> or <code>OutAttributes</code> is NULL. |
|-------------|---|

See also

`Is_ServerGetAttrs`.

Clients

SSM.

Notes

If the LS is told to shutdown, halt or reinitialize, this function returns before the operation completes.

9.2.9. Is_StatGetAttrs

Purpose

Retrieve Location Server statistics.

Syntax

```
#include "Is_interface.h"
```

```
signed32
```

```
Is_StatGetAttrs(  
    handle_t             Binding,      /* IN */  
    hpss_connect_handle_t *Connect,   /* IN */  
    Is_server_stats_t    *StatData)   /* OUT */
```

Description

The Is_StatGetAttrs function retrieves the runtime statistics of the Location Server. The caller must have control permissions on the Location Server's Security ACL in order to call this function.

Parameters

| | |
|-----------------|---|
| <i>Binding</i> | RPC binding handle to Location Server to contact. |
| <i>Connect</i> | Optional connection handle. |
| <i>StatData</i> | Returned server attributes. |

Return values

Upon successful completion, Is_StatGetAttrs returns zero. Otherwise, it returns a common error condition or one of the following error conditions described below.

Error conditions

| | |
|-------------|-------------------|
| HPSS_EFAULT | StatData is NULL. |
|-------------|-------------------|

See also

None.

Clients

SSM.

Notes

None.

9.2.10. Is_StatSetAttrs

Purpose

Set Location Server statistics.

Syntax

```
#include "Is_interface.h"
```

```
signed32
```

```
Is_StatSetAttrs(  
    handle_t                Binding,           /* IN */  
    hpss_connect_handle_t  *Connect,         /* IN */  
    u_signed64             InSelectBitmap,    /* IN */  
    u_signed64             *OutSelectBitmap,  /* OUT */  
    Is_server_stats_t     *InStatData,       /* IN */  
    Is_server_stats_t     *OutStatData)      /* OUT */
```

Description

The Is_StatSetAttrs function sets the runtime server statistics of the Location Server. The caller must have control permissions on the Location Server's Security ACL in order to call this function.

Parameters

| | |
|------------------------|--|
| <i>Binding</i> | RPC binding handle to Location Server to contact. |
| <i>Connect</i> | Optional connection handle. |
| <i>InSelectBitmap</i> | Bitmap of statistics fields to set. |
| <i>OutSelectBitmap</i> | Returned bitmap of statistics fields actually set. |
| <i>InStatData</i> | Statistics to set. |
| <i>OutStatData</i> | Returned statistics actually set. |

Return values

Upon successful completion, Is_StatSetAttrs returns zero. Otherwise, it returns a common error condition or one of the following error conditions described below.

Error conditions

HPSS_EFAULT OutSelectBitmap, InStatData or OutStatData is NULL.

See also

None.

Clients

SSM.

Notes

None.

9.3. Data Definitions

This section describes key internal data definitions and all externally used data definitions which are provided by this subsystem. A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

9.3.1. **Location Map Structure – ls_map_t**

Description

The Location Map Structure “maps” a single HPSS server’s object UUID to its DCE CDS Pathname. The various Location Map lookup APIs return this information for the server desired. This structure is mainly used by the client to connect locate a server to connect to.

Format

The ls_map_t structure has the following format:

```
typedef struct ls_map {
    uuid_t          UUID;
    uuid_t          HPSSId;
    char            CDSPath[HPSS_MAX_DCE_NAME];
    unsigned32     ServerType;
    unsigned32     Flags;
} ls_map_t;
```

UUID

The UUID of the HPSS server.

HPSSId

The HPSS identifier for the HPSS system that the server belongs to.

CDSPath

The CDS path needed to contact this HPSS server.

ServerType

The type of HPSS server. Valid values are:

| | |
|-----------------------|---------------------|
| BFS_SERVER_TYPE | Bitfile Server |
| DFSID_SERVER_TYPEDMAP | Gateway Server |
| DMG_SERVER_TYPE | DMAP Gateway Server |
| LS_SERVER_TYPE | Location Server |
| NS_SERVER_TYPE | Name Server |

Flags

Flags field for this HPSS server. Valid values are:

| | |
|-----------------|---|
| LS_MAP_IS_LOCAL | Server is in the local HPSS system. |
| LS_MAP_PERM | This map is a permanent entry. Client cache library only. |

LS_MAP_ROOT_NS Server is the root name server for the HPSS system.

LS_MAP_RPCGROUP Map represents a DCE RPC group of Location Servers.

Clients

The following clients access the data definition:

- Client API, Client Cache Library.

9.3.2. Location Map Array – ls_map_array_t

Description

The Location Map Array structure contains a set of location maps in conformant array format. This structure is used when Location Servers exchange remote map information.

Format

```
typedef struct ls_map_array {
    unsigned32      Count;
    [size_is(Count)] ls_map_t  Loc[*];
} ls_map_array_t;
```

Count

Number of location maps allocated to Loc.

Loc

Array of location maps.

Clients

The following clients access the data definition:

- Client API, Client Cache Library.

9.3.3. COS/BFS Selection Structure – ls_cos_bfs_t

Description

The COS/BFS Selection Structure represents a single Class of Server and Bitfile Server pairing. This pairing is used by the Client API while performing COS selection when a file is initially created.

Format

```
typedef struct ls_cos_bfs {
    uuid_t          BFSId;
    unsigned32      COSId;
    signed32        COSWeight;
    unsigned32      MaxOpenFiles;
    u_signed64      Bitfiles;
    u_signed64      FreeSpace;
    u_signed64      UsedSpace;
    u_signed64      BlockSize;
} ls_cos_bfs_t;
```

COSId

Class of Service (COS) ID.

BFSId

The UUID of the Bitfile Server.

COSWeight

The weight given to this COS based on COS Hints. This field is only useful when compared to other entries in the COS/BFS array passed back by the `hpss_LocateBFSByCOSHints` or `ls_BFSByCOSHints` call. The higher the value, the better the match.

MaxOpenFiles

The maximum number of file that this BFS may have open at the same time.

Bitfiles

The total number of bitfiles stored for this BFS. This value should be treated as an estimate. It will be correct for a specific point in time in the recent past.

FreeSpace

The amount of free space in bytes for this COS at this BFS. This value should be treated as an estimate. It will be correct for a specific point in time in the recent past.

UsedSpace

The amount of used space in bytes for this COS at this BFS. This value should be treated as an estimate. It will be correct for a specific point in time in the recent past.

BlockSize

Allocation block size in bytes for this COS at this BFS.

Clients

The following clients access the data definition:

- Client API, Client Cache Library.

9.3.4. COS/BFS Array Structure – `ls_cos_bfs_array_t`**Description**

The COS/BFS Array Structure is a conformant array of COS/BFS Selection structures. It is used by the Client API to determine COS and BFS selection during file creation.

Format

```
typedef struct ls_cos_bfs_array {
    unsigned32      Count;
    [size_is(Count)] ls_cos_bfs_t  *CosBfs;
} ls_cos_bfs_array_t;
```

Count

Number of COS/BFS pairs allocated to by `CosBfs`.

CosBfs

Conformant array of COS/BFS information.

Clients

The following clients access the data definition:

- Client API, Client Cache Library.

9.3.5. Location Server Statistics Structure – `ls_server_stats_t`

Description

The Location Server Statistics Structure contains runtime statistics for the Location Server. It is return to and reset by the SSM.

Format

```
typedef struct ls_server_stats {
    unsigned32      RegisterBitmap;
    unsigned32      RequestsPerMinute;
    unsigned32      RequestErrors;
    unsigned32      CosBfsRequests;
    unsigned32      CosBfsRemoteReqs;
    unsigned32      CosBfsUpdates;
    unsigned32      CosBfsUpdTimeouts;
    unsigned32      MinCosBfsUpdate;
    unsigned32      AvgCosBfsUpdate;
    unsigned32      MaxCosBfsUpdate;
    unsigned32      LocMapRequests;
    unsigned32      LocMapUpdates;
    unsigned32      LocMapUpdTimeouts;
    unsigned32      MinLocMapUpdate;
    unsigned32      AvgLocMapUpdate;
    unsigned32      MaxLocMapUpdate;
} ls_server_stats_t;
```

RegisterBitmap

A bitmap representing which fields SSM has currently registered for. These may be any of the following OR'd together:

| | |
|-------------------------|-------------------------------------|
| LS_STAT_REGISTERBITMAP | Represents RegisterBitmap field. |
| LS_STAT_REQPERMIN | Represents RequestsPerMinute field. |
| LS_STAT_REQERRORS | Represents RequestErrors field. |
| LS_STAT_COSBFSREQ | Represents CosBfsRequests field. |
| LS_STAT_COSBFSREMOTEREQ | Represents CosBfsRemoteReqs field. |
| LS_STAT_COSBFSUPD | Represents CosBfsUpdates field. |
| LS_STAT_COSBFSTIMEOUTS | Represents CosBfsUpdTimeouts field. |
| LS_STAT_MINCOSBFSUPD | Represents MinCosBfsUpdate field. |
| LS_STAT_AVGCOSBFSUPD | Represents AvgCosBfsUpdate field. |
| LS_STAT_MAXCOSBFSUPD | Represents MaxCosBfsUpdate field. |
| LS_STAT_LOCMAPREQ | Represents LocMapRequests field. |
| LS_STAT_LOCMAPUPD | Represents LocMapUpdates field. |
| LS_STAT_LOCMAPTIMEOUTS | Represents LocMapUpdTimeouts field. |
| LS_STAT_MINLOCMAPUPD | Represents MinLocMapUpdate field. |
| LS_STAT_AVGLOCMAPUPD | Represents AvgLocMapUpdate field. |

LS_STAT_MAXLOCMAPUPD Represents MaxLocMapUpdate field.

RequestsPerMinute

The number of CosBfs and LocMap requests processed during the previous minute.

RequestErrors

The total number of CosBfs and LocMap request errors returned to clients.

CosBfsRequests

The total number of CosBfs client requests received.

CosBfsRemoteReqs

The number of COS/BFS forwarded requests received from a remote Location Server.

CosBfsUpdates

The number of times COS/BFS statistics have been updated in the background.

CosBfsUpdTimeouts

The number of times a single BFS did not return COS/BFS statistics within the COS/BFS update interval.

MinCosBfsUpdate

The shortest time an update took to receive COS/BFS statistics from all BFSs. Timed out requests are not counted.

AvgCosBfsUpdate

The average time updates are taking to receive COS/BFS statistics from all BFSs. Timed out requests are not counted.

MaxCosBfsUpdate

The longest time an update took to receive COS/BFS statistics from all BFSs. Timed out requests are not counted.

LocMapRequests

The total number of Location map information client requests received.

LocMapUpdates

The number of times Location map information has been updated in the background.

LocMapUpdTimeouts

The number of times a single remote Location Server did not return Location map information within the Location map update interval.

MinLocMapUpdate

The shortest time an update took to receive Location map information from all remote Location Servers. Timed out requests are not counted.

AvgLocMapUpdate

The average time updates are taking to receive Location map information from all remote Location Servers. Timed out requests are not counted.

MaxLocMapUpdate

The longest time an update took to receive Location map information from all remote Location Servers.

Timed out requests are not counted.

Clients

The following clients access the data definition:

- SSM.

9.3.6. Location Policy Metadata Structure – `ls_policy_md_t`

Description

The Location Policy Metadata Structure contains information on how all of the local Location Servers will behave. It is similar to a server specific metadata record, but is shared by all local Location Servers.

Format

```
typedef struct ls_policy_md {
    unsigned32    Id;
    unsigned32    Pad1;
    char          CosMdFilename[HPSS_MAX_DCE_NAME];
    char          SiteMdFilename[HPSS_MAX_DCE_NAME];
    char          GroupName[HPSS_MAX_DCE_NAME];
    unsigned32    CosBfsUpdInterval;
    unsigned32    LocMapUpdInterval;
    unsigned32    MaxRequestThreads;
    unsigned32    MaxCosBfsThreads;
    unsigned32    MaxLocMapThreads;
    unsigned32    CosBfsTimeout;
    unsigned32    LocMapTimeout;
    unsigned32    Pad2;
    uuid_t        HPSSId;
} ls_policy_md_t;
```

Id

The index of this metadata record. This is always 1.

Pad1

An unused (reserved) field. Used for padding the metadata record properly.

CosMdFilename

The name of the COS metadata file.

SiteMdFilename

The name of the remote site metadata file.

GroupName

The name of the DCE RPC Group name used by clients to connect to the local Location Servers. Location Servers insert their CDS pathnames into this group during startup and remove them when shutting down.

CosBfsUpdInterval

The amount of time to wait, in seconds, between periodic background updates of COS/BFS information.

LocMapUpdInterval

The amount of time to wait, in seconds, between periodic background updates of location map information.

MaxRequestThreads

The maximum number of threads to use to allow to handle incoming client RPC requests. This value plus the values in `MaxCosBfsThreads` and `MaxLocMapThreads` should not be greater than 500.

MaxCosBfsThreads

The maximum number of threads to use to gather COS/BFS statistic information. In any case, the actual number of threads used will not exceed the number of active local BFSs.

MaxLocMapThreads

The maximum number of threads to use to gather location mapping information. In any case, the actual number of threads used will not exceed the number of active remote Location Servers.

CosBfsTimeout

The maximum amount of time to wait, in seconds, for BFS to return COS/BFS information.

LocMapTimeout

The maximum amount of time to wait, in seconds, for a remote Location Server to return location information.

Pad2

An unused (reserved) field. Used for padding the metadata record properly.

HPSSId

The unique identifier for this HPSS system (installation).

Clients

The following clients access the data definition:

- SSM.

9.3.7. Remote HPSS Site Metadata Structure – `hpss_site_md_t`

Description

The Remote HPSS Site Metadata Structure contains information on how to contact a Location Server located at a remote HPSS site. The information stored in this record contains information entered into the Location Policy Metadata Structure at the remote site. It is used by the local Location Servers to connect to and exchange information with remote Location Servers.

Format

```
typedef struct hpss_site_md {
    uid_t      HPSSId;
    char       DescName[HPSS_MAX_DESC_NAME];
    char       LSGroupName[HPSS_MAX_DCE_NAME];
    unsigned32 Pad1;
    unsigned32 Pad2;
} hpss_site_md_t;
```

HPSSId

The unique identifier for the remote HPSS system. This should be copied from the remote site's Location Policy Metadata Structure.

DescName

A descriptive name for the remote HPSS system.

LSGroupName

The DCE CDS pathname of the remote RPC group used to contact a remote Location Server. This should be copied from the remote site's Location Policy Metadata Structure. Make sure the value is fully qualified (i.e. it begins with `/.../<remote-cell's-name>`).

Pad1

An unused (reserved) field. Used for padding the metadata record properly.

Pad2

An unused (reserved) field. Used for padding the metadata record properly.

Clients

The following clients access the data definition:

- Location Server, SSM.

Appendix A - Acronyms

| | |
|--------|---|
| ACL | Access Control List |
| AIX | Advanced Interactive Executive |
| API | Application Program Interface |
| BFS | Bitfile Server |
| CCPI | Client Cache Programming Interface |
| CDS | Cell Directory Server |
| DCE | Distributed Computing Environment |
| DIR | Database Interface Routines |
| DOE | Department of Energy |
| EFS | External File System |
| FTP | File Transfer Protocol |
| GID | Group Identifier |
| HPNS | HPSS POSIX Name Server |
| HPSS | High Performance Storage System |
| IBM | International Business Machines Corporation |
| ID | Identifier |
| IEEE | Institute of Electrical and Electronics Engineers |
| I/O | Input / Output |
| IOD | I/O Descriptor |
| IOR | I/O Reply |
| IP | Internet Protocol |
| IPI | Intelligent Peripheral Interface |
| LANL | Los Alamos National Laboratory |
| LIR | Local Interface Routines |
| LLNL | Lawrence Livermore National Laboratory |
| LS | Location Server |
| MPI-IO | Message Passing Interface – Input / Output |
| MPS | Migration Purge Server |
| MSSRM | Mass Storage System Reference Model |
| NFS | Network File System |
| NS | Name Server |
| ORNL | Oak Ridge National Laboratory |
| PIOFS | Parallel I/O File System |
| POSIX | Portable Operating System Interface for computer environments (an IEEE operating system standard) |
| PVL | Physical Volume Library |
| PVR | Physical Volume Repository |
| RIR | Remote Interface Routines |
| RISC | Reduced Instruction Set Computer |
| RPC | Remote Procedure Call |
| RSN | Relative Sequence Number |
| SFS | Structured File Server |
| SIR | System Interface Routines |
| SNL | Sandia National Laboratories |
| SPI | Server Programming Interface |
| SS | Storage Segment Storage Server |
| SSM | Storage System Manager |
| SP | Scalable Processor |
| TCP | Transmission Control Protocol |
| UID | User Identifier |
| UUID | Universally Unique Identifier |

Appendix A

| | |
|-----|-----------------|
| VV | Virtual Volume |
| UID | User Identifier |

Appendix B - References

1. ***HPSS Error Messages Manual***, April 1999.
2. ***HPSS Programmer's Reference Guide, Volume 1***, April 1999.
3. ***HPSS System Administration Guide***, April 1999.
4. ***HPSS User's Guide***, April 1999.
5. ***Institute of Electrical and Electronics Engineers (IEEE) Mass Storage System Reference Model (MSSRM) (Version 5)***.

