HPSS 10.3.0.0.0 Programmer’s Reference

Generated on Fri Sep 29 2023

Contents

1 Introduction 1

2 Preface 3

2.1 Copyright notification . . . 3

2.2 Trademark usage . . . 3

3 Overview 5

3.1 Client API . . . 5

3.1.1 Purpose . . . 5

3.1.2 Components . . . 5

3.1.3 Constraints . . . 6

3.1.4 Libraries . . . 6

3.1.5 User-defined Attributes Search APIs . . . 6

3.1.6 HPSS Checksum . . . 7

3.1.7 PIO Model . . . 7

3.1.8 Trashcans . . . 8

3.2 64-bit Arithmetic Library . . . 8

3.2.1 Purpose . . . 9

3.2.2 Constraints . . . 9

3.2.3 Libraries . . . 9

3.3 Storage Concepts . . . 9

3.3.1 Class of Service . . . 9

3.3.2 Storage Class . . . 10

3.3.3 Storage Hierarchy . . . 10

3.3.4 File Family . . . 10

3.3.5 User IDs . . . 10

3.4 Access Control List API . . . 10

3.4.1 Purpose . . . 10

3.4.2 Constraints 11
3.4.3 Libraries 11

4 Client API Tutorials and Best Practices 13

4.1 Client API Concepts 13
4.1.1 How the Client API Works 13
4.1.2 Client Platforms and Considerations 13
4.1.3 Limitations and Known Issues 14
4.1.4 Standard Practices 14
4.1.5 HPSS Levels 14
4.2 Installation and Configuration 14
4.2.1 Installation 14
4.2.2 Install from RPM 14
4.2.3 Install from Source 15
4.2.4 Using pkg-config To Compile 16
4.2.5 Testing 16
4.2.6 Configuration 16
4.2.7 Client API Environment Variables 16
4.2.8 Client API Runtime Configuration 19
4.2.9 Other System Configuration 19
4.3 Tuning and Troubleshooting 19
4.3.1 Tuning 19
4.3.2 Expectations 19
4.3.3 Testing Procedures 20
4.3.4 Tuning Concepts 20
4.3.5 Troubleshooting 21
4.4 Client API Uses 22
4.4.1 Overview and Applications 23
4.5 Tutorials 23
4.5.1 Overview 23
4.5.2 A Short Example: Current Working Directory 23
4.5.3 Namespace Metadata 26
4.5.4 File Input and Output 31
4.5.5 HSM Functions 35
4.5.6 User-defined Attributes (UDA) 38

4.5.7 Using UDAs with VFS 44
4.5.8 HPSS to POSIX and Back Again 44
4.5.9 More On UDAs 45
4.5.10 Useful References 47

5 Developer Acknowledgements **49**

6 Further Reading **51**

7 Glossary of Terms and Acronyms **53**

8 Deprecated List **61**

9 Module Documentation **63**

9.1 File hash flag values 63
9.1.1 Detailed Description 63
9.1.2 Variable Documentation 63
9.2 Client API 65
9.2.1 Detailed Description 71
9.2.2 Class Documentation 71
9.2.3 Macro Definition Documentation 79
9.2.4 Typedef Documentation 82
9.2.5 Enumeration Type Documentation 83
9.2.6 Function Documentation 85
9.3 File Open, Create, and Close 113
9.3.1 Detailed Description 113
9.3.2 Function Documentation 114
9.4 File Data 125
9.4.1 Detailed Description 126
9.4.2 Function Documentation 126
9.5 Fileset and Junction 137
9.5.1 Detailed Description 137
9.5.2 Function Documentation 137
9.6 File Attribute 145
9.6.1 Detailed Description 146
9.6.2 Function Documentation 146
9.7 File Name 165

9.7.1 Detailed Description 165
9.7.2 Function Documentation 165
9.8 ACLs 170
9.8.1 Detailed Description 170
9.8.2 Function Documentation 170
9.9 Directory Creation and Deletion 176
9.9.1 Detailed Description 176
9.9.2 Function Documentation 176
9.10 Directory Data 180
9.10.1 Detailed Description 180
9.10.2 Function Documentation 181
9.11 Working Directory 189
9.11.1 Detailed Description 189
9.11.2 Function Documentation 189
9.12 Client API Control 192
9.12.1 Detailed Description 192
9.12.2 Function Documentation 192
9.13 Authentication 198
9.13.1 Detailed Description 198
9.13.2 Function Documentation 198
9.14 HPSS Statistics 201
9.14.1 Detailed Description 201
9.14.2 Function Documentation 201
9.15 HPSS Object 205
9.15.1 Detailed Description 205
9.15.2 Function Documentation 205
9.16 Buffered Data 208
9.16.1 Detailed Description 208
9.16.2 Function Documentation 208
9.17 User-defined Attributes 217
9.17.1 Detailed Description 218
9.17.2 Function Documentation 218
9.18 Data Structure Conversion 229
9.18.1 Detailed Description 229

9.18.2 Function Documentation 229
9.19 Parallel I/O 232
9.19.1 Detailed Description 232
9.19.2 Function Documentation 232
9.20 Trashcan 238
9.20.1 Detailed Description 238
9.20.2 Function Documentation 238
9.21 HPSS Environment Variable Module 243
9.21.1 Detailed Description 243
9.21.2 Function Documentation 243
9.22 HPSS Networking Library 245
9.22.1 Detailed Description 245
9.22.2 Function Documentation 246
9.23 HPSS XML Interface 255
9.23.1 Detailed Description 255
9.23.2 Function Documentation 255
9.24 64-bit Math Library 263
9.24.1 Detailed Description 264
9.24.2 Macro Definition Documentation 264
9.25 HPSS Client API Extensions 269
9.25.1 Detailed Description 269
9.25.2 Macro Definition Documentation 269
9.25.3 Function Documentation 270
9.26 HPSS Config File Template API 275
9.26.1 Detailed Description 275
9.27 Main 276
9.27.1 Detailed Description 276
9.27.2 Function Documentation 276
9.28 Parsers 278
9.28.1 Detailed Description 278
9.28.2 Function Documentation 279
9.29 Special 280
9.29.1 Detailed Description 280
9.30 Validators 281

9.30.1 Detailed Description 281
9.31 POSIX APIs 282
9.31.1 Detailed Description 282
9.31.2 Function Documentation 282
9.32 Data Containers 287
9.32.1 Detailed Description 290
9.32.2 Class Documentation 290
9.32.3 Macro Definition Documentation 291
9.32.4 Typedef Documentation 291
9.32.5 Enumeration Type Documentation 293
9.32.6 Function Documentation 293
9.33 Hash Interface 310
9.33.1 Detailed Description 311
9.33.2 Macro Definition Documentation 311
9.33.3 Function Documentation 311
9.34 HPSS Config File API 319
9.34.1 Detailed Description 321
9.34.2 Class Documentation 321
9.34.3 Macro Definition Documentation 322
9.34.4 Typedef Documentation 322
9.34.5 Enumeration Type Documentation 322
9.34.6 Function Documentation 322
9.35 HPSS Access Control List APIs 326
9.35.1 Detailed Description 327
9.35.2 Class Documentation 327
9.35.3 Macro Definition Documentation 327
9.35.4 Typedef Documentation 328
9.35.5 Function Documentation 328
9.36 Gatekeeper Site Library 334
9.36.1 Detailed Description 334
9.36.2 Class Documentation 335
9.36.3 Typedef Documentation 335
9.36.4 Function Documentation 336
9.37 Account Validation Site Library 342

9.37.1 Detailed Description 342
9.37.2 Function Documentation 342

10 Class Documentation 351

10.1 hpss_Attrs Struct Reference 351
10.1.1 Detailed Description 352
10.1.2 Member Data Documentation 352

11 File Documentation 357

11.1 acct_Site.c File Reference 357
11.2 acct_site.h File Reference 358
11.3 api_access.c File Reference 358
11.3.1 Detailed Description 359
11.4 api_acct.c File Reference 359
11.4.1 Detailed Description 360
11.4.2 Function Documentation 360
11.5 api_acl.c File Reference 362
11.5.1 Detailed Description 362
11.6 api_cache.c File Reference 363
11.6.1 Detailed Description 364
11.6.2 Function Documentation 364
11.7 api_chdir.c File Reference 370
11.7.1 Detailed Description 370
11.8 api_chmod.c File Reference 370
11.8.1 Detailed Description 370
11.9 api_chown.c File Reference 371
11.9.1 Detailed Description 371
11.10 api_chroot.c File Reference 371
11.10.1 Detailed Description 371
11.10.2 Function Documentation 371
11.11 api_close.c File Reference 373
11.11.1 Detailed Description 373
11.12 api_closedir.c File Reference 373
11.12.1 Detailed Description 373
11.13 api_copyfile.c File Reference 374

11.13.1 Detailed Description 374
11.14 api_cos.c File Reference 374
11.14.1 Detailed Description 374
11.15 api_distfile.c File Reference 374
11.15.1 Detailed Description 375
11.16 api_fgetattr.c File Reference 375
11.16.1 Detailed Description 376
11.16.2 Function Documentation 376
11.17 api_fileset.c File Reference 379
11.17.1 Detailed Description 380
11.18 api_fsetattr.c File Reference 380
11.18.1 Detailed Description 381
11.19 api_getcwd.c File Reference 381
11.19.1 Detailed Description 381
11.20 api_gettrash.c File Reference 381
11.20.1 Detailed Description 382
11.21 api_ids.c File Reference 382
11.21.1 Detailed Description 382
11.21.2 Function Documentation 382
11.22 api_init.c File Reference 383
11.22.1 Detailed Description 385
11.22.2 Function Documentation 385
11.22.3 Variable Documentation 390
11.23 api_junction.c File Reference 390
11.23.1 Detailed Description 391
11.24 api_link.c File Reference 391
11.24.1 Detailed Description 391
11.25 api_lookup.c File Reference 391
11.25.1 Detailed Description 392
11.26 api_lseek.c File Reference 392
11.26.1 Detailed Description 392
11.27 api_misc.c File Reference 392
11.27.1 Detailed Description 396
11.27.2 Function Documentation 396

11.28 api_misc_admin.c File Reference 419
11.28.1 Detailed Description 419
11.29 api_mkdir.c File Reference 419
11.29.1 Detailed Description 419
11.30 api_open.c File Reference 420
11.30.1 Detailed Description 420
11.31 api_opendir.c File Reference 421
11.31.1 Detailed Description 421
11.32 api_path.c File Reference 421
11.32.1 Detailed Description 422
11.32.2 Function Documentation 422
11.33 api_pio.c File Reference 426
11.33.1 Detailed Description 427
11.33.2 Function Documentation 427
11.34 api_posix_lseek.c File Reference 427
11.35 api_posix_open.c File Reference 428
11.36 api_purge.c File Reference 428
11.36.1 Detailed Description 429
11.37 api_purgelock.c File Reference 429
11.37.1 Detailed Description 429
11.38 api_purgeonmigrate.c File Reference 429
11.38.1 Detailed Description 429
11.39 api_rddir.c File Reference 429
11.39.1 Detailed Description 430
11.39.2 Function Documentation 431
11.40 api_rdlink.c File Reference 431
11.40.1 Detailed Description 431
11.40.2 Function Documentation 432
11.41 api_rdlist.c File Reference 433
11.41.1 Detailed Description 433
11.41.2 Function Documentation 433
11.42 api_read.c File Reference 434
11.42.1 Detailed Description 435
11.43 api_register.c File Reference 435

11.43.1 Detailed Description 435
11.44 api_rename.c File Reference 435
11.44.1 Detailed Description 435
11.45 api_rewdir.c File Reference 436
11.45.1 Detailed Description 436
11.46 api_rmdir.c File Reference 436
11.46.1 Detailed Description 437
11.46.2 Function Documentation 437
11.47 api_stage.c File Reference 438
11.47.1 Detailed Description 439
11.47.2 Function Documentation 439
11.48 api_stat.c File Reference 442
11.48.1 Detailed Description 442
11.49 api_statfs.c File Reference 442
11.49.1 Detailed Description 443
11.50 api_statsubsys.c File Reference 443
11.50.1 Detailed Description 443
11.51 api_stream.c File Reference 443
11.51.1 Detailed Description 444
11.52 api_symlink.c File Reference 444
11.52.1 Detailed Description 445
11.53 api_trunc.c File Reference 445
11.53.1 Detailed Description 445
11.54 api_umask.c File Reference 445
11.54.1 Detailed Description 446
11.55 api_undelete.c File Reference 446
11.55.1 Detailed Description 446
11.56 api_unlink.c File Reference 446
11.56.1 Detailed Description 447
11.56.2 Function Documentation 447
11.57 api_userattr.c File Reference 447
11.57.1 Detailed Description 449
11.58 api_utime.c File Reference 449
11.58.1 Detailed Description 449

11.59 api_wrappers.c File Reference 449
11.59.1 Detailed Description 450
11.59.2 Function Documentation 450
11.60 api_write.c File Reference 450
11.60.1 Detailed Description 451
11.61 api_wrtlist.c File Reference 451
11.61.1 Detailed Description 452
11.61.2 Function Documentation 452
11.62 bfs_userif_def.x File Reference 452
11.62.1 Detailed Description 453
11.62.2 Class Documentation 453
11.62.3 Variable Documentation 457
11.63 core_api_def.x File Reference 457
11.63.1 Detailed Description 458
11.63.2 Class Documentation 458
11.63.3 Enumeration Type Documentation 460
11.63.4 Variable Documentation 461
11.64 gk_gk_IFStructs.x File Reference 461
11.64.1 Detailed Description 461
11.64.2 Class Documentation 461
11.64.3 Enumeration Type Documentation 462
11.65 gk_RTM.h File Reference 462
11.66 gk_SiteProtos.h File Reference 463
11.67 hacl.h File Reference 463
11.68 hpss_api.h File Reference 465
11.68.1 Class Documentation 480
11.68.2 Typedef Documentation 481
11.68.3 Enumeration Type Documentation 481
11.68.4 Function Documentation 482
11.69 hpss_api_def.x File Reference 489
11.69.1 Detailed Description 490
11.69.2 Class Documentation 490
11.70 hpss_cos.x File Reference 490
11.70.1 Detailed Description 490

11.70.2 Class Documentation 491
11.71 hpss_dirent.h File Reference 494
11.71.1 Detailed Description 494
11.71.2 Class Documentation 495
11.72 hpss_Getenv.c File Reference 495
11.72.1 Detailed Description 496
11.72.2 Function Documentation 496
11.73 hpss_Getenv.h File Reference 496
11.73.1 Function Documentation 496
11.74 hpss_hash.c File Reference 497
11.74.1 Detailed Description 498
11.74.2 Macro Definition Documentation 498
11.75 hpss_hash.h File Reference 498
11.76 hpss_ihashtable.h File Reference 500
11.76.1 Detailed Description 500
11.76.2 Typedef Documentation 500
11.77 hpss_ilist.h File Reference 500
11.77.1 Detailed Description 502
11.77.2 Typedef Documentation 504
11.77.3 Enumeration Type Documentation 504
11.78 hpss_irbtree.c File Reference 504
11.78.1 Detailed Description 506
11.79 hpss_irbtree.h File Reference 506
11.79.1 Detailed Description 508
11.80 hpss_Mech.c File Reference 508
11.80.1 Detailed Description 509
11.80.2 Function Documentation 509
11.81 hpss_net.c File Reference 509
11.81.1 Detailed Description 510
11.81.2 Function Documentation 510
11.82 hpss_net.h File Reference 513
11.82.1 Function Documentation 514
11.83 hpss_posix_api.h File Reference 516
11.83.1 Detailed Description 517

11.84 hpss_sec_api.x File Reference . . . 517
11.84.1 Detailed Description . . . 517
11.84.2 Class Documentation . . . 517
11.85 hpss_stat.h File Reference . . . 518
11.86 hpss_String.c File Reference . . . 518
11.86.1 Detailed Description . . . 522
11.86.2 Function Documentation . . . 522
11.86.3 Variable Documentation . . . 537
11.87 hpss_TrashRecord.x File Reference . . . 544
11.87.1 Class Documentation . . . 544
11.88 hpss_types.x File Reference . . . 545
11.88.1 Detailed Description . . . 546
11.88.2 Class Documentation . . . 546
11.88.3 Macro Definition Documentation . . . 547
11.88.4 Typedef Documentation . . . 547
11.88.5 Enumeration Type Documentation . . . 548
11.88.6 Variable Documentation . . . 550
11.89 hpss_vattr.h File Reference . . . 550
11.89.1 Detailed Description . . . 551
11.90 hpss_xml.c File Reference . . . 551
11.90.1 Detailed Description . . . 552
11.91 hpss_xml.h File Reference . . . 552
11.92 hpsscfg_config_api.c File Reference . . . 552
11.92.1 Detailed Description . . . 553
11.93 hpsscfgx_cfg_functions.c File Reference . . . 553
11.93.1 Detailed Description . . . 554
11.94 hpsscfgx_GetClientInterfaces.c File Reference . . . 554
11.94.1 Detailed Description . . . 554
11.95 hpsscfgx_hpssconf.c File Reference . . . 554
11.95.1 Detailed Description . . . 555
11.96 hpsscfgx_pattern_match.c File Reference . . . 555
11.96.1 Detailed Description . . . 556
11.96.2 Macro Definition Documentation . . . 556
11.97 hpsscfgx_restricted_ports.c File Reference . . . 556

11.97.1 Detailed Description 557
11.98 hpssclntaddr.x File Reference 557
11.98.1 Detailed Description 557
11.98.2 Class Documentation 557
11.99 ns_interface_defs.x File Reference 557
11.99.1 Detailed Description 558
11.99.2 Class Documentation 558
11.99.3 Typedef Documentation 563
11.100 ns_ObjHandle.x File Reference 563
11.100.1 Detailed Description 563
11.100.2 Class Documentation 563
11.101 ss_pvlist.x File Reference 564
11.101.1 Detailed Description 564
11.101.2 Class Documentation 564
11.102 u_signed64.h File Reference 565
11.102.1 Macro Definition Documentation 566

Index **567**

Chapter 1

Introduction

The *HPSS Programmer's Reference* describes areas of HPSS which can be used in custom or site applications.

Preface Contains contractual and trademark information.

Overview The Overview contains a brief description of the HPSS Client API and general concepts of programming for HPSS.

Client API Tutorials and Best Practices Client API Best Practices The Best Practices Guide contains a tutorial-style discussion of the HPSS Client API with examples.

API Reference Links:

- Client API Describes available data structures and functions for the Client API.
- HPSS Networking Library Describes available data structures and functions for the HPSS network library.
- Hash Interface Describes available data structures and functions for the HPSS hashing library.
- HPSS XML Interface Describes available data structures and functions for the HPSS XML module.
- HPSS Environment Variable Module Describes available data structures and functions for the HPSS Environment module.
- 64-bit Math Library Describes available data structures and functions for the HPSS 64-bit math library.
- Gatekeeper Site Library Describes available data structures and functions for the HPSS Gatekeeper Site library.
- Account Validation Site Library Describes available data structures and functions for the HPSS Accounting Site library.
- HPSS Access Control List APIs Describes available data structures and functions for the HPSS ACL Library.
- HPSS Client API Extensions Describes available data structures and functions for the HPSS Configuration Extensions.

Glossary of Terms and Acronyms A brief glossary of technical terms and acronyms.

Further Reading Additional reading, papers, specifications, etc.

Developer Acknowledgements Developer Acknowledgements

The *HPSS Programmer's Reference* is generated from HPSS source and should generally be accurate, but sometimes mistakes sneak through, spelling errors happen, or additional information may be useful. Please contact HPSS support if you find errors in this reference.

Chapter 2

Preface

2.1 Copyright notification

2.2 Trademark usage

Chapter 3

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 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 are the programming interfaces for accessing data from HPSS.

3.1 Client API

3.1.1 Purpose

The purpose of the Client API is to provide an interface which mirrors the POSIX.1 specification where possible to provide ease of use for the POSIX application programmer. In addition, extensions are provided to the programmer to take advantage of the specific features provided by HPSS, such as storage/access hints passed on file creation, parallel data transfers, migration, and purge.

Consult the Client API Tutorials and Best Practices for a discussion of programming with the Client API.

3.1.2 Components

The Client API consists of these major parts:

- Authentication
- File Open, Create, and Close
- File Data
- Buffered Data
- Parallel I/O
- Directory Creation and Deletion
- Directory Data
- File Attribute
- File Name
- Working Directory

- Data Structure Conversion
- User-defined Attributes
- Fileset and Junction
- ACLs
- HPSS Object
- HPSS Statistics
- Client API Control
- Trashcan

3.1.3 Constraints

The following constraints are imposed by the Client API:

- The validity of open files and directories at the time of fork is undefined in the child process.
- The validity of open files and directories is lost across calls to any of the family of exec calls.
- The designed client API works only with applications that make use of the Core Server. In particular, this API is not designed to meet the needs of clients that will perform the Name Service functionality internally or will bypass the Core Server when performing storage operations.
- The client API does not support the file locking interfaces (for example, fcntl(2) & flock(2)) found in most UNIX operating systems. Because of this, it is the responsibility of the application to prevent interceding update problems by handling any necessary serialization.
- Multithreaded applications should take care to serialize calls to exit(). Per the ISO C standard, multiple calls to exit() are undefined.

3.1.4 Libraries

Applications issuing HPSS Client API calls must link the following library:

libhpss.a HPSS client library

Applications issuing requests to the HPSS POSIX API must link the following library:

libhpssposix.a HPSS POSIX client library

3.1.5 User-defined Attributes Search APIs

Searching should only be done on user-defined attributes which have an index defined over them. Indexes can be defined on the system running DB2 using the xmladdindex utility. Additionally, because of speed concerns when searching, if the XML for each result is not required, the hpss_UserAttrGetObjQuery() interface should be used instead of the hpss_UserAttrGetXMLObj() interface.

3.1.6 HPSS Checksum

HPSS checksum functions provide a method of error detection. Changes in a file can be detected; however, neither the location of the change nor the amount of change can be determined from the checksum. Checksumming does not enable recovery from data errors. Mirrored copies, RAIT, or other error correcting mechanisms are required to recover from data.

The checksum functions work by generating a checksum engine context for the chosen algorithm. That engine context is then used by several generic functions to append information, finalize the hash, reset the engine context, or free the engine context.

To use the HPSS checksum functions, follow the following model:

1. Authenticate with HPSS
2. Open a file
3. Create a hash engine (hpss_HashCreateXXX)
4. Read the file and append each data buffer to the Hash Engine (hpss_HashAppend(), hpss_HashAppendStr())
5. Close the file.
6. Finalize the checksum (hpss_HashFinish() or hpss_HashFinishHex())
7. At this point you have a finished checksum which can be used at a later time to determine if the contents of the file have changed.

The following algorithms are supported: SHA1, SHA224, SHA256, SHA384, SHA512, MD5, Crc32, and Adler32.

3.1.7 PIO Model

A description of a simple programming model for PIO is provided in this section. The PIO functions use a programming model of one coordinator and one or many participants. The coordinator organizes and manages the data transfer, while the participants collect the data. Here is an outline of a simple PIO program:

1. Authenticate with HPSS
2. Open a file
3. (Coordinator) hpss_PIOStart()
4. (Coordinator) hpss_PIOExportGrp()
5. Spawn participants
6. (Participants) hpss_PIOImportGrp()
7. (Participants) hpss_PIORegister()
8. (Participants) hpss_PIOFinalize()
9. (Coordinator) hpss_PIOExecute()
10. (Coordinator) hpss_PIOFinalize()

Note the dual role of hpss_PIOFinalize(). It is used to clean up the participants and also to complete the I/O and clean up the coordinator. This is a very simple overview of how to structure a PIO program, but it is accurate in the roles and order in which the functions need to be called to get PIO to function properly.

3.1.8 Trashcans

Trashcans (also called "soft delete") in HPSS are currently enabled or disabled on a system-wide basis. In general, applications should not need to consider whether trashcans are on or off when doing deletion; the functionality is transparent to the application.

When trashcans are enabled, object deletion is considered "soft". Any delete operation will place the object into a trashcan. The objects in this trashcan will then eventually be deleted by the system based upon a system-wide policy. A delete on an object which is already in the trashcan permanently deletes it and frees up the associated space, while moving a file out of a trashcan will keep it from being deleted by the system, as will "undeleting" the file (see hpss_Undelete()).

Moving a file into the trash will not result in a soft deletion; the file must actually be deleted. When a file is deleted into a trashcan it is renamed to append its object ID onto the end of it. For example, myfile.out with object ID 1234 becomes myfile.out.1234. Similarly, moving a trashcan directory does not undelete its contents.

There are two types of trashcans: home directory and fileset. A typical user will have one home directory trashcan and one fileset trashcan per additional fileset. When a user deletes an object in the same fileset as their home directory, the system will move that object into the user's home directory trashcan. This is ∼/.Trash. When a user deletes an object in a fileset which differs from their home directory, or if the user has no existing home directory, the object will be placed in the user's fileset trash. This is [fileset root]/.Trash/[user name]. These trashcan directories are automatically created and do not need to be set up by the application. For example, in the root fileset with the hpss user it may be /.Trash/hpss. In a fileset under /sys1 it would be /sys1/.Trash/hpss. Files cannot be soft deleted across filesets.

3.1.8.1 Getting Information on Trashcans

Information about whether trashcanning is on or not, and additional settings, can be obtained by priviledged users using the hpss_GetTrashSettings() API. In general, developers should assume that the trashcan behavior could be changed by the administrator at any time.

Information about a trashcanned object's original location can be obtained using normal attribute functions. See the data structure reference for more infomation.

3.1.8.2 Limitations, Caveats, and Warnings

Currently, operations are not restricted within a trashcan; however it is not advised to develop applications which take any special advantage of this fact as operations on soft-deleted objects may be restricted later.

In its current implementation, HPSS trashcans are a flat directory. This means that while perusing a trashcan, it is impossible to determine the relationships between directories and files in the trashcan because all directories and files reside at the top of the trashcan directory. At this time there is no API which identifies the trashcan containing a deleted file or contents across multiple trashcans.

The Client API caches trashcan directories to reduce unnecessary lookups. In certain cases this may cause the Client API to issue a retry on a delete operation because the cache entry became invalid and had to be refreshed. In some cases a client instance may continue deleting files into a .Trash directory which has been moved elsewhere due to caching. The Client API will not discover that the .Trash has been moved until it is restarted. It is not recommended to move or alter trashcan directories.

3.2 64-bit Arithmetic Library

3.2.1 Purpose

Some HPSS Client APIs require 64-bit fields. The operating system and C compiler on many workstation platforms may not support 64-bit integer operations. As a result, in order to support large integer fields, a set of math libraries has been supplied until 64-bit support is available on all supported vendor platforms.

The Client API has been compiled and tested in 64-bit mode on the four supported platforms. This results in the 64-bit Arithmetic Library components being converted into true 64-bit operations.

3.2.2 Constraints

The following constraints are imposed by the 64-bit arithmetic functions:

- 64-bit signed arithmetic operations are not supported.
- Multiply functions are limited to 64-bit by 32-bit unsigned operations. For example, a 64-bit unsigned integer may be multiplied by a 32-bit unsigned integer. No 64-bit by 64-bit operations are supported for this category of functions.

3.2.3 Libraries

The 64-bit arithmetic functions are included in the **libhpss.a** library.

3.3 Storage Concepts

This section defines key HPSS storage concepts which have a significant impact on the usability of HPSS. Configuration of the HPSS storage objects and policies is the responsibility of your HPSS system administrator.

3.3.1 Class of Service

Class of Service (COS) is an abstraction of storage system characteristics that allows HPSS users to select a particular type of service based on performance, space, and functionality requirements. Each COS describes a desired service in terms of characteristics such as minimum and maximum file size, segment allocation method, transfer rate, access frequency, latency, and valid read or write operations. A file resides in a particular COS and the class is selected when the file is created. Underlying a COS is a storage hierarchy that describes how data for files in that class are to be stored in HPSS.

You specify a COS at file create time by using COS hints and priority structures which are passed to HPSS in the hpss_Open() function. You can later change the COS by using the hpss_FileSetCOS() or hpss_FileSetCOSHandle() functions. Contact your HPSS system administrator to determine the Classes of Service which have been defined. Use the following command to list the defined Classes of Service:

```
lshpss -cos
```

Refer to Chapter 4 of the *HPSS User's Guide* for information on the lshpss command. A class of service is implemented by a storage hierarchy of one to many storage classes. Storage hierarchies and storage classes are not directly visible to the user, but are described below since they map to a Class of Service. The relationship between storage class, storage hierarchy, and COS is shown in Figure 3-1.

3.3.2 Storage Class

An HPSS storage class is used to group storage media together to provide storage with specific characteristics for HPSS data. The attributes associated with a storage class are both physical and logical. Physical media in HPSS are called physical volumes. Physical characteristics associated with physical volumes are the media type, block size, the estimated amount of space on volumes in this class, and how often to write tape marks on the volume (for tape only). Physical media are organized into logical virtual volumes. This allows striping of physical volumes. Some of the logical attributes associated with the storage class are virtual volume block size, stripe width, data transfer rate, latency associated with devices supporting the physical media in this class, and storage segment size (disk only). In addition, the storage class has attributes that associate it with a particular migration policy and purge policy to help in managing the total space in the storage class.

3.3.3 Storage Hierarchy

An HPSS storage hierarchy consists of multiple levels of storage with each level representing a different storage media (i.e., a storage class). Files are moved up and down the storage hierarchy via stage and migrate operations, respectively, based upon storage policy, usage patterns, storage availability, and user request. For example, a storage hierarchy might consist of a fast disk, followed by a fast data transfer and medium storage capacity robot tape system, which in turn is followed by a large data storage capacity, but relatively slow data transfer tape robot system. Files are placed on a particular level in the hierarchy depending on the migration policy and staging operations. Multiple copies of a file may also be specified in the migration policy. If data is duplicated for a file at multiple levels in the hierarchy, the more recent data is at the higher level (lowest level number) in the hierarchy. Each hierarchy level is associated with a single storage class.

3.3.4 File Family

A file family is an attribute of an HPSS file that is used to group a set of files on a common set of tape virtual volumes. Grouping of files only on tape volumes is supported. Families can only be specified by associating a family with a fileset, and creating the file in the fileset. When a file is migrated from disk to tape, it is migrated to a tape virtual volume assigned to the family associated with the file. If no family is associated with the file, the file is migrated to the next available tape not associated with a family (actually to a tape associated with family zero). If no tape virtual volume is associated with the family, a blank tape is reassigned from family zero to the file's family. The family affiliation is preserved when tapes are repacked. Configuring file families is an HPSS System Administrator's function.

3.3.5 User IDs

After the HPSS system is configured, the necessary accounts must be created for HPSS users. Contact your HPSS system administrator to add an account. For the Client API, either a UNIX or Kerberos account must be created. The HPSS system administrator can use the following commands to add a new account.

```
hpssuser -add user -krb
hpssuser -add user -unix
```

3.4 Access Control List API

3.4.1 Purpose

The access control list API is a set of routines for managing access control lists (ACLs). The routines provide a way to convert ACLs from string format into a form suitable for use by the client API routines. They also provide a way to call the client API routines using ACLs and string format, and a way to convert ACLs back from client API format to string format. In particular, the string conversion routines take care of translating user, group, and realm names into UIDs, GIDs and Realm IDs respectively.

3.4.2 Constraints

None.

3.4.3 Libraries

The access control list APIs are available in the HPSS Client Library, **libhpss.a**.

Chapter 4

Client API Tutorials and Best Practices

This chapter provides guidance on using the HPSS client programming interface (Client API). In addition, short examples and snippets are provided to facilitate new development. Distinctions are made between the characteristics of various API functions which may differ from their typical file system counterparts.

4.1 Client API Concepts

The HPSS Client API provides both POSIX-like interfaces into the HPSS system as well as interfaces specific to managing hierarchical storage. Given the unique nature of the underlying storage system and its metadata, many functions vary from POSIX to some degree, which is why they are described as POSIX-like. HPSS itself is not POSIX conformant, it is POSIX compliant, which means that partial support for POSIX.1 is available, and the features which are supported are documented (in this document). You may assume that any POSIX features not explicitly described in this document are unavailable.

A number of variants of POSIX functions as well as additional functions are provided to allow the development of useful applications which can take advantage of the special features of a hierarchical storage management system, and in some cases these functions have characteristics which may differ from a typical file system implementation. In this section some of these special characteristics will be discussed as well as other more general characteristics of the HPSS Client API, hereafter referred to simply as the Client API or the API.

4.1.1 How the Client API Works

The Client API works by contacting the HPSS Location Server, retrieving the information needed to contact the Core Server, and then issuing requests to the Core Server. In environments with multiple Core Servers, the Location Server reports the location of each server, and then the Client API detects the correct Core Server for each request based upon either a user-input subsystem id or the subsystem id of the request's object handle. The Core Server handles all requests from the API, with I/O requests being handled by the Mover. The Client connects to the Mover, or the Mover to the client, depending upon the type of I/O, with the Core Server handling the passing of initial connection information. The Core Server does not directly transfer any data. Core Server requests are made via RPC, and data connections to the Mover are made via direct TCP/IP connection.

4.1.2 Client Platforms and Considerations

Consult your HPSS version release notes for proper platform and library prerequisites for the Client API.

4.1.3 Limitations and Known Issues

1. HPSS does not support links which cross junctions.
2. The Client API's default open file limit is 4096. This is a per process limitation, and the Core Server has its own (configurable) limitation.
3. The **HPSS_API_DISABLE_JUNCTIONS** flag must currently be set to '4' in order to work properly.
4. Running the Client API under some memory checking programs such as Valgrind may result in a large number of undefined value errors due to undefined values passed in HPSS RPC interfaces as outputs.
5. Purging a login credential with hpss_PurgeLoginCred() will fail with an error if a login has already been established.

4.1.4 Standard Practices

Many applications choose to spawn a separate process for each Client API I/O operation. Threaded usage of the API is also supported. Having separate processes has a few advantages such as reducing the chance of hitting the open file limitations.

4.1.5 HPSS Levels

Some applications may wish to define certain actions based on the **HPSS_LEVEL** macro. It is a five-digit number built using the major, minor, patch, and build levels of HPSS:

```
(10000*HPSS major level)+(1000*HPSS minor level)+(100*HPSS maintenance level)+HPSS patch level
```

For example, the HPSS 7.4.2 release has **HPSS_LEVEL** of 74200, and HPSS 7.4.1 Patch 3 has **HPSS_LEVEL** of 74103.

4.2 Installation and Configuration

4.2.1 Installation

This section will discuss the necessary steps required to install the Client API. It is important to rebuild your application client binaries following any upgrade as there is no guarantee of backwards compatibility with older versions of HPSS. Failure to do this can result in undefined behavior, failures, or program crashes due to incompatibilities between the HPSS client and server. It is important to thoroughly test your application prior to running it in production with a new version of HPSS. This is especially true when moving up several versions or doing a major upgrade.

4.2.2 Install from RPM

Installing the HPSS Client API from RPM requires identifying the proper client RPM. RPMs are available for all supported architectures and operating systems.

4.2.2.1 Client API Binary Package

The hpss-clnt RPM installs HPSS Client API tool binaries.

```
% rpm -i hpss-clnt-<version>.<platform>.<arch>.rpm
```

Either the hpss-lib or hpss-clnt-devel RPM must be installed as prerequisite for hpss-clnt.

4.2.2.2 Client API Development Package

The hpss-clnt-devel RPM installs HPSS Client API libraries and headers.

```
% rpm -i hpss-clnt-devel-<version>.<platform>.<arch>.rpm
```

4.2.2.3 Client API Mover Protocol Development Package

The hpss-clnt-devel-mvrprot RPM installs HPSS Client API headers for implementing mover protocol.

Warning

> The headers in the mvrprot RPM are considered confidential intellectual property and may not be distributed for any reason without approval from IBM. These headers are available to HPSS licensees upon request.

```
% rpm -i hpss-clnt-devel-mvrprot-<version>.<platform>.<arch>.rpm
```

4.2.3 Install from Source

Normally, any install on a supported platform will use the RPM install process (Install from RPM). The install from source method should only be used by special arrangement.

The Client API source is distributed along with all other HPSS source code. In order to build the Client API on a client machine, it is necessary to extract it and compile it. This is called the client bundle.

4.2.3.1 Extraction from HPSS Source Tree

In order to build the Client API it first must be extracted from the HPSS source tree along with its dependencies and client tools. This is accomplished with a single make command issued from the root of an HPSS source bundle.

```
% make build-clnt BUILD_ROOT=/path/to/client/build
```

This will copy the necessary files to the specified location. Keep in mind that any Makefile customization specific to that source tree will be retained when the files are copied; for instance, the build platform, HPSS root path, library or include locations, etc. These may need to be modified once the bundle is moved to the client environment.

4.2.3.2 Compilation

Compiling the Client API is straightforward, but there are several Makefile options which may need to be modified depending upon the environment. The primary Makefile is Makefile.macros, and it has several important flags and values which may need to be changed depending on your desired configuration.

```
BUILD_PLATFORM = LINUX
```

Set the BUILD_PLATFORM to the O/S the client will run on. The legal values are AIX, SOLARIS, and LINUX. This value in turn sets a number of variables based upon defaults found in the Makefile.macros.[OS] Makefiles. The system-specific defaults are not typically changed, but can control things such as library locations and binaries for use in the Makefiles.

```
BUILD_TOP_ROOT = /opt/hpss
```

This is the HPSS source directory. It is highly recommended that this be a link to some other area so that multiple source code directories can coexist and be maintained without confusion, and if necessary the client source can be backed up to an older version.

```
BIT64_SUPPORT = on
```

This flag determines whether the Client API is built with 32-bit or 64-bit binaries. Finally, within the root of the HPSS source tree, run:

```
% make clnt
```

This builds the Client API, its dependencies, and some Client API tools (most notably **scrub**). You should at a minimum see the HPSS client libraries in **$HPSS_ROOT/lib**: **libhpss.so** and **libhpsscs.so**. Depending upon your security settings you will see **libhpssunixauth.so** and possibly **libhpsskrb5auth.so** as well.

4.2.4 Using pkg-config To Compile

Both the source and RPM installs of the Client API include a package config file. This file can be used to assist in compiling your application with the HPSS Client API by providing a set of compile flags and library dependencies which are commonly required for HPSS applications. The pkgconfig file is located in: **${HPSS_ROOT}/lib/pkgconfig/hpss.pc**.

Note that if the install path is not the final path for the libraries and headers, the 'prefix' path in hpss.pc can be modified to allow the continued use of pkgconfig.

Note also that the pkgconfig file is only valid for the platform associated with the RPM. In order to use the Client API on a different platform, architecture, operating system, or bit-width, the RPM for that specific environment must be obtained and installed.

4.2.5 Testing

There are a several tools and utilities which are either included with the HPSS software package or available as support tools which can test the Client API. The first and foremost Client API test utility is **scrub**. **scrub** provides a simple shell interface which can be used to test connectivity, namespace operations, file I/O, and much more. The **hpsssum** utility is also included in the client build, and can be used to read existing files out of HPSS or generate / modify file checksum metadata stored with the UDAs feature.

4.2.6 Configuration

Client API settings can be configured in three ways: environment variables, at run-time, and through other HPSS and system files. All HPSS environment variables and the most common alternate configurations will be described.

4.2.7 Client API Environment Variables

Each of these environment variables is defined in hpss_env_defs.h, which is appended onto the HPSS Management Guide, and are also described in other reference material such as the *HPSS Programmer's Reference*. For the sake of completeness this guide will provide a few short examples of practical applications of each of these environment variables.

HPSS_API_HOSTNAME

This is the data hostname that the API will use when supplying an address to the Mover when initiating a transfer. Typically this is used to specify an alternate hostname which maps to a high speed interconnect, while the host will continue to use a hostname mapped to a slower control network when communicating with the Core Server. It is important to note the Mover must be able to connect to this alternate interface.

HPSS_API_DEBUG / HPSS_API_DEBUG_PATH

The **HPSS_API_DEBUG** and **HPSS_API_DEBUG_PATH** environment variables are described in some detail in the Troubleshooting section; they control the level of detail and output path of API logging.

HPSS_API_MAX_CONN This value is currently unused; the total maximum connections across all Client API applications can be configured using the Location Server and Core Server.

HPSS_API_MAX_OPEN The Client API instance open file limit is configurable. The default value is 4096 open files. Applications may increase this by setting an environment variable, **HPSS_API_MAX_OPEN**. This may be set either in the env.conf or in the application environment. Hitting this limit will cause further opens to fail. The lower limit for the configured Client API maximum open files is 1; the upper limit for the configured Client API maximum open files is (2^31-1).

HPSS_API_RETRIES

This value limits the total number of retries which will be done by the Client API where the class of the error is "-Retryable". "Retryable" errors will be retried right away. Retries can be turned off by setting this to '-1'. The following Core Server errors are considered retryable:

- HPSS_EINPROGRESS, -EINPROGRESS
- HPSS_ETIMEDOUT
- HPSS_EBUSY
- HPSS_ENOTREADY

HPSS_API_BUSY_DELAY

This value is the time (in seconds) to wait between retries when the class of error is "delay retryable". The following Core Server errors are considered delay retryable:

- HPSS_ENOCONN
- HPSS_BUSY_RETRY (only applies when manipulating a bitfile, e.g., open, close, stage)
- BFS_STAGE_IN_PROGRESS (if API_RetryStageInp is turned on)

HPSS_API_TOTAL_DELAY

This value is the total time (in seconds) across all retries that the request may be retried.

HPSS_API_LIMITED_RETRIES

This value is the number of times to retry when the class of error is "limited retryable". This class of retryable operations will be retried immediately with no delay a limited number of times. The following Core Server errors are considered limited retryable:

- HPSS_EPIPE
- HPSS_ECONN
- HPSS_EBADCONN

HPSS_API_DMAP_WRITE_UPDATES

This environment variable is defunct and has been removed from the API.

- HPSS_API_REUSE_CONNECTIONS

If this option is enabled the Client API will request that storage resources be held until the connection is closed, and reuse data connections.

HPSS_API_USE_PORT_RANGE

This value causes the Client API to request that the Mover connect back using the port range specified in the mover port range configuration. However this value has no meaning when PDATA PUSH is used because the Client connects to the Mover rather than the Mover connecting back to the Client.

HPSS_API_RETRY_STAGE_INP

This value, when nonzero, will interpret the BFS_STAGE_IN_PROGRESS error as retryable. This translates into errors caused by staging being retried rather than immediately returning with an error. This is on by default.

HPSS_API_DISABLE_CROSS_REALM

This is used to reject requests to initialize the Client API when one of the realms is not local. Cross-realm support is a special configuration of HPSS which allows HPSS installations to communicate, and is not generally useful for most customers.

HPSS_API_DISABLE_JUNCTIONS

This option will disable the crossing of junctions by the Client API. This essentially limits any client applications to the root fileset and a single subsystem, and can be useful in restricting access.

HPSS_API_SITE_NAME

This value is not used by the Client API, but may be useful for client applications which connect to multiple HPSS systems.

HPSS_API_AUTHN_MECH

This option will set the default authentication mechanism used by the Client API. Applications can specify a different mechanism at run time. In sites which use both Unix and Kerberos this can be useful in defaulting a certain set of clients to one authentication mechanism or the other.

HPSS_API_RPC_PROT_LEVEL

This value controls the RPC protection level of the API connections made to the Core and Location Servers. The effects of these values are documented in the HPSS Installation Guide; the allowed values are: connect packet, packet integrity, and packet privacy.

HPSS_API_SAN3P

This environment variable determines whether the Client API will use SAN3P. In certain circumstances it may be beneficial to turn this off and use the network interface for data transfer instead; for example, the SAN interface is down or a certain client may not need access to the disk resources on the SAN.

HPSS_API_TRANSFER_TYPE

This is the default transfer type which will be used by the API. The three values are TCP, MVRSELECT, and IPI. IPI is defunct, TCP implicitly disables SAN3P, and MVRSELECT allows SAN3P (if available).

HPSS_API_OBJ_BUF_LIMIT

This represents the maximum number of objects which can be returned by a single hpss_UserAttrGetObjs() or hpss_UserAttrReadObjs() operation. The Core Server will adhere to this limit when handling the UDAs calls which retrieve Object Ids. This is meant to be an administrative limitation placed on all API clients at the Core Server layer; modifying this value on Client machines has no effect.

HPSS_API_XML_BUF_LIMIT

This represents the maximum number of XML strings which can be returned by a single hpss_UserAttrGetXML() or hpss_UserAttrReadXML() operation. The Core Server will adhere to this limit when handling the UDAs calls which retrieve XML strings. This is meant to be an administrative limitation placed on all API clients at the Core Server layer; modifying this value on Client machines has no effect.

HPSS_API_XMLOBJ_BUF_LIMIT

This represents the maximum number of objects which can be returned by a single hpss_UserAttrGetXMLObj() or hpss_UserAttrReadXMLObj() operation. The Core Server will adhere to this limit when handling the UDAs calls which retrieve Object Id and XML string pairs. This is meant to be an administrative limitation placed on all API clients at the Core Server layer; modifying this value on Client machines has no effect.

HPSS_API_XMLSIZE_LIMIT

This represents the maximum size of a single XML string which will be returned by an hpss_UserAttrGetXML(), hpss_UserAttrGetXMLObj(), hpss_UserAttrReadXML(), or hpss_UserAttrReadXMLObj() request. Any operation requesting strings longer than this limit will be rejected. This is meant to limit memory consumption by the Core Server as it is possible to request XML strings up to 2 GB (provided appropriate infrastructure exists).

HPSS_API_XMLREQUEST_LIMIT

This represents the maximum total size of a single request returning XML strings. Any operation with a total request size longer than this limit will be rejected. The total request size is calculated as (size of request strings $*$ number of request strings). This is meant to limit memory consumption by the Core Server as it is possible to request multiple XML strings, each having a very large size.

4.2.8 Client API Runtime Configuration

Most of the environment variables enumerated above correspond to a configurable field or flag within the Client API Configuration data structure (see hpss_api.h). These fields and flags can be retrieved from the API by using the hpss_GetConfiguration() function, and can be set using the hpss_SetConfiguration() function. This allows a more finely grained and manageable configuration that does not necessarily rely upon external configuration files which may be shared between different applications.

4.2.9 Other System Configuration

Some applications have their own configuration files which change Client API behavior when used within that application. For example, the HPSS.conf file is used by FTP to allow user configuration of many parameters, including parameters related to the Client API.

4.3 Tuning and Troubleshooting

4.3.1 Tuning

Tuning is as much art as science, and in general almost no specific tuning parameter is a best fit for everything. This section describes what kinds of things impact the performance of the Client API.

4.3.2 Expectations

Some technologies within HPSS may have some surprising characteristics for new users. Application programmers should be aware of some of the characteristics of these technologies so that they can write applications that utilize the strengths of the underlying technology.

4.3.2.1 Tape

Tape is a sequential I/O technology with relatively high seek times to get to a random position on tape. Tape has the additional characteristic of not being "always on" like a disk - a tape cartridge must first be mounted into a tape drive before it can be read, and also a tape drive has a start-up time before it can reach its maximum I/O rate.

HPSS systems make heavy use of tape storage for long-term archival. It is important to recognize when constructing an application that data on tape may not be readily available due to factors like location of the cartridge and availability of tape drives - a read request from tape could take minutes or hours between the time the request is generated and the time the first byte is received. Additionally, assuming that an application is reading directly from tape, a short read of one part of a file followed by a short read of another part of a file may cause a noticeable delay

4.3.2.2 Metadata

Metadata is data about the data. It may describe where it's located, how it's laid out on media, who can access it, etc. In HPSS, metadata is located in a relational database. Some of the characteristics of a relational database apply to the access and retrieval of HPSS metadata. For example, a simple directory listing typically translates into an SQL select, and multiple reads from the database. Using this knowledge, it should come as no surprise to learn that a short delay could be expected before the first results appear in a very large directory. After all, the relational database will fetch all the results when the SQL is run, and the time required to do that is dependent upon the number of entries.

4.3.3 Testing Procedures

It is important to test that the Client API deployment was successful. HPSS support maintains a repository of example programs which can be used to test functionality, and the **scrub** tool can also be used to test functionality. It is appropriate to test both namespace and I/O operations. When testing client applications, consider how the HPSS APIs you are using are impacting performance. Doing this frequently can help identify possible performance issues early. Perhaps there's a better API for the task.

4.3.4 Tuning Concepts

The important information to know are what you're trying to optimize, and some common things that can be done to optimize those things. Depending upon specific deployments and specific hardware the specific steps and choke-points may be completely different; however the optimization points are still the same. The Client API will perform best when:

1. A reliable connection exists between the machines running the Client API, the Movers, and the Core Server machines. Avoid public WANs which can drop long-running connections.
2. The Location Server and Core Server are configured in anticipation of the planned client and system administrative load. The server Maximum Requests size should be larger than the anticipated number of clients to accommodate administrative programs, such as various tools including **repack**.
3. High speed interconnects are used between the Client API and Mover. The API data hostname should be configured properly to take advantage of high speed interconnections. For SAN3P, it's important to verify that the SAN3P disks are usable by the client.
4. Storage class definitions are appropriate for the data being transferred. Misconfigured storage classes can lead to network and storage inefficiencies due to large numbers of protocol messages and transactions. I/O performance is sensitive to storage segment sizes due to an HPSS internal limit of 32 descriptors in a request at a time.
5. Parallelism. HPSS is meant to scale very well at the data layer. Multiple client machines and many client processes is considered normal. Application developers should expect to use multiple processes in order to get maximum performance.

6. Software Tuning. Poorly written software can have a significant detrimental impact to overall application performance. Some of the most common issues are inappropriate buffer sizes, misuse of APIs, or making assumptions about performance of various HPSS APIs based upon past experience programming for file systems. For example, simple namespace operations such as renaming or creating a hardlink is more expensive in HPSS than a typical file system.

4.3.5 Troubleshooting

HPSS and the Client API provide various ways of troubleshooting Client API applications. Among these are configurable logging subsystems on the HPSS servers and within the Client API as well as system files for the security subsystem. This section will describe various methods and issues that may arise during troubleshooting.

4.3.5.1 Debug and Trace Levels

HPSS has eight types of logging which can be configured per server. When doing API development it is often useful to turn REQUEST and TRACE logging on for the Location Server and Core Server in order to more easily identify where certain errors come from. Additional logging on other servers (for example, the Mover) may be appropriate depending upon the nature of the error. The Client API has three types of debugging which can be controlled through the **HPSS_API_DEBUG** environment variable: ERROR, REQUEST, and TRACE. These are controlled by an octal value in the API Logs section below. Within the context of the Client API, ERROR logs are due to any error code being returned by the Core Server - this can include so-called 'informative error codes' which do not result in operational failure. REQUEST logs are application requests to the Client API, and TRACE level will log information during normal operations describing API activities or state values. Other logs mentioned typically have no configurable logging level; that is, they are either on or off.

4.3.5.2 API Logs

The HPSS API logs are a very useful first step in troubleshooting HPSS Client API problems. They can be turned on within the shell by setting the **HPSS_API_DEBUG** environment variable, by the application (using hpss_SetAPILogLevel() / hpss_SetAPILogPath()), or by using the HPSS API logging resource file.

Below is a table which indicates which logging types will be enabled based upon the value of **HPSS_API_DEBUG**. This debug output will go to stdout by default, and can be redirected by specifying an alternate path using the **HPSS_API_DEBUG_PATH** environment variable.

Value	Trace	Request	Error
0	Off	Off	Off
1	Off	Off	On
2	Off	On	Off
3	Off	On	On
4	On	Off	Off
5	On	Off	On
6	On	On	Off
7	On	On	On

Consult the Client API function references for hpss_SetAPILogLevel() and hpss_SetAPILogPath() for details on setting the log level and path programmatically. These log level / path updates can be done while the application is running post-initialization.

The HPSS API Resource file allows system-level manipulation of HPSS Client API logging, overriding application-supplied log parameters. This file must be created as:

${HPSS_PATH_TMP}/hpss.api.resource (e.g. /var/hpss/tmp/hpss.api.resource)

Once the file exists and has valid log level and path information, all Client API logging will redirect there after the applications detect the change (within 30 seconds). If the application cannot read the resource file due to file or directory permissions, then the resource file cannot be used.

The format of the file is:

```
<log level> <log path>
```

The log path has a limit of 1023 characters and the log level is the same as the table above. The new logging will persist until the resource file is removed, after which time the log levels and paths will return to their original levels and locations. Remember that this resource file controls output for the entire sytem on which it is placed, so logs can become large very quickly. The resource file cannot be disabled; it must be removed in order for resource-based log levels to be removed.

It is worth noting that some error log entries within the API may be expected and may not constitute a hard error. For example, on initialization the API will attempt to place the authenticated user in their home directory. If that home directory does not exist you may see an error log even though the application will continue to function. Other error messages are logged but serve as informational errors from the Core Server back to the API, and provide useful information for understanding the program flow. An example of this is an API error called **HPSS_EMOREPATH** which will be returned and logged whenever a junction is crossed. In general the API debug log should only be turned on when troubleshooting a specific problem; the logging mechanism has no roll-over mechanism and so the log file can grow without bound.

4.3.5.3 HPSS Logs

Sometimes the information provided in the API log is not enough to understand what went wrong. At that point it is useful to start looking through the HPSS log messages. It is possible to trace a request from the Client API through the HPSS logs using the Request Id. This Request Id is presented in the API log as a hex value, for example:

```
trc:02/27/2012 14:22:18.389320[0x5f500001]: entering API_TraversePath: Path=/,Handle=0xf7f5fac0
```

0x5f500001 is the Request Id as a hex string; the decimal representation which will be logged by the Core Server is 1599078401. If you have an API log which shows an error being returned from the Core Server you can easily find the HPSS logs which correspond to it by converting the hex string and finding that Request Id (1599078401) in the HPSS log. The Request Ids generated by the API are very likely, but not guaranteed, to be unique within a time window of hours or even days, depending upon the level of system activity.

Certain errors, such as authentication errors, are not always logged by the HPSS server, or the details needed to resolve the problem aren't logged in all cases. In order to get additional information about authentication and authorization related issues, use the /var/hpss/tmp/gasapi.diag and /var/hpss/tmp/secapi.diag files. These files can be created on the client machine or the Core Server to begin logging security and authentication information, and they will log information about every authentication request. These files do not timestamp or trim themselves and so they should be turned on just long enough to debug the issue. In all cases it's necessary to stop each process which was using the log files before the space is truly freed - for example all clients must be stopped on the client side, and all servers must be stopped on the Core Server side. This can cause service disruption and so its typically desirable to plan the use of these log files around a window where a small outage would be acceptable.

4.3.5.4 Crash Dumps

Occasionally you may get an application crash due to a software or hardware error. In such cases, you should use the logs and the generated core file to get an understanding of what caused the crash. There's nothing special about debugging the Client API in a core dump. If the problem appears to originate within the Client API, contact HPSS support with the core dump trace and a description of what the application was doing so that the issue can be analyzed.

4.4 Client API Uses

4.4.1 Overview and Applications

The Client API is the most general interface into HPSS. Every client interface packaged with HPSS (PFTP, VFS) and many other interfaces use the Client API as a lower layer.

PFTP and VFS have very different deployment models for the Client API. PFTP runs a daemon on the HPSS Core Server, and requests from PFTP clients funnel through that daemon, which in turn uses the Client API. This allows PFTP to be deployed with a minimum of prerequisite software and configuration data. VFS, on the other hand, deploys the Client API on every VFS client machine, which includes all of its libraries and configuration files.

4.5 Tutorials

This section provides examples and discussion of the most-used APIs including an in-depth tutorial showing a complete view of creating an API program and compiling it from scratch. Additional tutorials will discuss small snippets centered around single functions or logical groups of functions.

4.5.1 Overview

The following tutorial will provide a thorough examination of the creation of a Client API program from start to finish. Each additional tutorial focuses on a small topic such as creating a file and present a snippet of code which accomplishes that task. Those tutorials also discuss related issues and common variations on the task. There are over 200 APIs in the HPSS Client API, and not all of them can be covered in this level of detail, therefore only the APIs most likely to be used are covered by this document. The Client Interfaces section of this document does provide a basic description of every API, its parameters, and its expected return codes. These tutorials should not be misinterpreted as a replacement for the API references for understanding specific Client API functions.

4.5.2 A Short Example: Current Working Directory

Assumptions: This tutorial (and other tutorials in this section) should be run on the Core Server as it uses HPSS system user IDs to log in. It may be run on a properly configured client system if the HPSS system account is created there, or if the program is modified to use an account that is available on both the Core Server and the client host. This tutorial assumes that a directory with the compiled HPSS client source exists, and we'll refer to it as **HPSS_ROOT**. It will also assume, in the case a client machine is used, that the prerequisite configuration files have been copied over and the Client API has been successfully tested for connectivity and working configuration with the Core Server and the Mover using **scrub** or a similar utility. Consider the following simple program:

```c
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <hpss_errno.h>
#include <hpss_api.h>
#include <hpss_Getenv.h>
#include <hpss_limits.h>

int
main(int argc, char** argv)
{
  int rc;
  char cwd_path[HPSS_MAX_PATH_NAME];

  char keytab_file[HPSS_MAX_PATH_NAME] = "/var/hpss/etc/hpss.unix.keytab";

  unsigned char* env, *keytab;
  hpss_authn_mech_t mech;
  int i;

  env = hpss_Getenv("HPSS_PRIMARY_AUTHN_MECH");

  if(env != NULL) {
     if(strcasecmp(env, "UNIX") == 0) {
        mech=hpss_authn_mech_unix;
        env = (char*)hpss_Getenv("HPSS_UNIX_KEYTAB_FILE");
        strcpy(keytab_file, env);
       }
       else if(strcasecmp(env,"KRB5") == 0) {
          mech=hpss_authn_mech_krb5;
          env = (char*)hpss_Getenv("HPSS_KRB5_KEYTAB_FILE");
          strcpy(keytab_file, env);
       }
       else {
          printf("Unknown authentication mechanism.  Defaulting to Unix.\n");
          mech=hpss_authn_mech_unix;
       }

    // remove auth_keytab prefix
    for(i = 0; i < strlen(keytab_file); i++)
       if(keytab_file[i] == ':')
         break;
    i++;
    keytab=keytab_file+i;

  }
  else {
      printf("Primary authentication not defined.\n");
      exit(1);
  }



  if((rc = hpss_SetLoginCred("hpssssm",
                             mech,
                             hpss_rpc_cred_client,
                             hpss_rpc_auth_type_keytab,
                             keytab)) < 0)
  {
      printf("could not authenticate.\n");
      exit(1);
  }


  rc = hpss_Getcwd(cwd_path, HPSS_MAX_PATH_NAME);

  printf("Path is %s\n", cwd_path);

  return 0;
}
```

The following gcc command will successfully compile this Client API program on Linux:

```
gcc `pkg-config --cflags --libs /opt/hpss.75/lib/hpss.pc` -c hpss_test_program.c
gcc `pkg-config --cflags --libs /opt/hpss.75/lib/hpss.pc` hpss_test_program.o -o ../bin/hpss_test_program
```

This is done in a two step process (compile and link). For simple programs the two can easily be combined in gcc. HPSS provides a pkgconfig file for the Client API which is based upon the environment the Client API was compiled in. By default it is in the $HPSS_ROOT/lib folder as 'hpss.pc'. It includes a number of useful pieces of information such as the HPSS version, the compile flags, and the libraries required.

Users of the client bundle (client source) will generate a new pkgconfig file when the tree is compiled. Users of the build-clnt-devel bundle (binaries and headers only) will receive a pkgconfig file with the bundle.

The pkgconfig file can be customized to some extent by the developer as necessary. pkgconfig files are not guaranteed to be compatible across HPSS releases.

Now, let's discuss the headers in the example above:

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <hpss_errno.h>
#include <hpss_api.h>
#include <hpss_Getenv.h>
#include <hpss_limits.h>
```

The HPSS error codes are contained in hpss_errno.h. HPSS error codes are defined as negative numbers, and try to conform to their errno equivalent where possible. For example, HPSS error code HPSS_EIO is -5; Linux error code EIO is 5. The hpss_api.h header defines the data structures and function prototypes for the HPSS Client API. The hpss_Getenv.h defines the hpss_Getenv() function, which is used to retrieve HPSS environment variables from the default list, the override file, or the environment. Finally, hpss_limits.h defines various limits within HPSS including the max path size.

```
int
main(int argc, char** argv)
{
  int rc;
  char cwd_path[HPSS_MAX_PATH_NAME];

  char keytab_file[1024] = "/var/hpss/etc/hpss.unix.keytab";

  unsigned char* env, *keytab;
  hpss_authn_mech_t mech;
  int i;
```

This section defines the main function and creates some local variables which will be used later. It is worth noting that HPSS_MAX_PATH_NAME is used here to hold the current working directory path and the keytab file name - using HPSS_MAX_PATH_NAME for HPSS paths is generally a good idea - and a variable of hpss_authn_mech_t type will be used to hold the authentication mechanism.

```
env = hpss_Getenv("HPSS_PRIMARY_AUTHN_MECH");

if(env != NULL) {
   if(strcasecmp(env, "UNIX") == 0) {
      mech=hpss_authn_mech_unix;
      env = (char*)hpss_Getenv("HPSS_UNIX_KEYTAB_FILE");
      strcpy(keytab_file, env);
     }
     else if(strcasecmp(env,"KRB5") == 0) {
        mech=hpss_authn_mech_krb5;
        env = (char*)hpss_Getenv("HPSS_KRB5_KEYTAB_FILE");
        strcpy(keytab_file, env);
     }
     else {
        printf("Unknown authentication mechanism.  Defaulting to Unix.\n");
        mech=hpss_authn_mech_unix;
     }

  // remove auth_keytab prefix
  for(i = 0; i < strlen(keytab_file); i++)
     if(keytab_file[i] == ':')
       break;
  i++;
  keytab=keytab_file+i;

}
else {
    printf("Primary authentication not defined.\n");
    exit(1);
}
```

This section obtains the default authentication mechanism and the associated keytab. This is useful because we're using keytab authentication for this program. It's also possible to use a password, and also possible to create your own HPSS keytab file with non-HPSS system users. hpss_Getenv() pulls its environment variables from the environment , the overrides file, and finally a set of default defined values, and will prefer them in that order - that is, if a variable is set in the environment, hpss_Getenv() will use that, and then check the overrides file, and finally the defaults.

```
if((rc = hpss_SetLoginCred("hpssssm",
                           mech,
                           hpss_rpc_cred_client,
                           hpss_rpc_auth_type_keytab,
                           keytab)) < 0)
{
    printf("could not authenticate.\n");
    exit(1);
}
```

Now we finally get to do something. The hpss_SetLoginCred() function logs into HPSS and creates a security credential with the information we provide. The user "hpssssm" can be useful for testing purposes, or you can substitute your own user who has an entry in the appropriate keytab file. This user should exist on the system that the program is running on as well as the HPSS Core Server. Using a user which already exists on the test system is done to make this tutorial shorter and simpler, but in general your application should use your own user rather than an HPSS system user name. Logging in with a non-system user name is done in the same way, although you may have to provide a password or a custom keytab rather than the HPSS default keytab.

```
    rc = hpss_Getcwd(cwd_path, HPSS_MAX_PATH_NAME);
    printf("Path is %s\n", cwd_path);
    return 0;
}
```

Since we're now logged in it's possible to use the Client API. Prior to this the application would have seen errors such as HPSS_EPERM (-1) when attempting to access HPSS through the Client API. Our simple tutorial application merely calls hpss_Getcwd(), which returns the current working directory. In this case your application may print "/" or "/home/hpssssm" depending upon what the hpssssm user's home directory is set to. If you change the program and run with a different user you'll likely see the current working directory as that user's home directory, unless it has not been created. Client API initialization will happen the first time the Client API is called after logging in, and creates the settings the API will run under (for instance, the environment variables described in the Client API Environment Variables section).

The remainder of the tutorials within this document will contain only the information necessary to understand the usage of each API rather than a full program. This tutorial program can be used as a base from which to test and manipulate these other snippets. The code required to log into HPSS described above will be moved to a function called "login" for brevity.

4.5.3 Namespace Metadata

The HPSS namespace presents a way to navigate and organize files within directories. Every file within HPSS has an entry in the namespace. In order to create a file, it must be have namespace metadata associated with it. The following tutorials focus on creating, retrieving, and manipulating namespace metadata. The characteristics of the namespace are such that a working mover is not required in order to satisfy its operations; all of the namespace metadata resides on the Core Server.

4.5.3.1 Create

The following snippet will create a new HPSS file, assuming that appropriate permissions exist. There are variants of this function such as hpss_Create(), hpss_Creat(), and others which provide alternative methods for creating and optionally opening an HPSS file.

```
int     hfd;
hpss_cos_hints_t  hints_in, hints_out;
hpss_cos_priorities_t  hints_pri;
char *hfile;

login();

memset(&hints_in, 0x0, sizeof(hints_in));
memset(&hints_out, 0x0, sizeof(hints_out));
memset(&hints_pri, 0x0, sizeof(hints_pri));

hfile=argv[1];

hfd = hpss_Open(hfile, O_CREAT, 0777,
                &hints_in, &hints_pri, &hints_out);
if (hfd < 0)
{
    fprintf(stderr, "Could not open/create: %s, error %d\n", hfile,             hfd);
    exit(1);
}

hpss_Close(hfd);
```

The hints are an important addition to the open API. They allow the caller to specify several criteria and requirements for the HPSS Class of Service (COS) that will be used to store the new file's data. One common usage is to simply require a specific Class of Service be used, and another may require a certain File Family be used. The resulting COSId will be presented via the hints_out parameter in the example above.

There are a few other notable features of hpss_Open(). One of them is that the file remains open until it is closed, and the HPSS Client API has a file table limit defined by **HPSS_API_MAX_OPEN** open files at a single time. The default is 4096 open files. Across multiple instances, each instance could have up to **HPSS_API_MAX_OPEN** files open with an appropriately configured HPSS Location and Core Server. hpss_Open() also allows the mode permissions to be supplied, however it is important to note that when testing, the umask will be applied to these mode bits; use hpss_Umask() to change the umask settings.

4.5.3.2 Rename

The following snippet will rename an existing HPSS file, assuming that appropriate permissions exist for both the old and new locations. There are other ways to do this operation, for example by creating and removing hard links.

```
int  rc;
char *oldfile, *newfile;

login();

oldfile=argv[1];
newfile=argv[2];

rc = hpss_Rename(oldfile, newfile);
if (rc < 0)
{
    fprintf(stderr, "Could not rename: %s, error %d\n", oldfile, rc);
    exit(1);
}
```

The rename operation is very simple; one thing to watch out for troubleshooting rename issues is to be sure that both the old and new areas should be checked for proper conditions when a rename fails.

4.5.3.3 Unlink

The following snippet will demonstrate the hpss_Unlink() function. The hpss_Unlink() function is similar to the unlink(2) function in that it will always result in a namespace entry being removed, but may or may not actually remove any data. For example, regular files will have their data removed, but a hardlink may not remove any data if additional hardlinks to that bitfile exist. Unlinks on a directory will succeed only if the directory is empty - to recursively delete directory contents, it's necessary to traverse the directory subtrees and empty the directories before deleting them. Unlinks on symbolic links will likewise not delete any file data.

```
int  rc;
char *file;

login();

file=argv[1];

rc = hpss_Unlink(file);
if (rc < 0)
{
    fprintf(stderr, "Could not unlink: %s, error %d\n", file, rc);
    exit(1);
}
```

Unlinks in HPSS are performed in two stages where in the first stage the namespace object is removed, and in the second the bitfile is removed. The bitfile is placed on a list of files to be removed and will be cleaned up as allowed by the system. The upshot from this is if you are deleting a large number of files, or a number of heavily used files, you may not see space being freed for some time even though the namespace entries have been removed.

4.5.3.4 Get Attributes

This snippet will retrieve file attributes for a namespace entry. This can be used, for example, to retrieve the number of entries in a directory, the size of a file, the last time the file was written to, and more:

```
int  rc;
char *file;
hpss_fileattr_t attrs;

login();

memset(&attrs, 0x0, sizeof(attrs));

file=argv[1];

rc = hpss_FileGetAttributes(file, &attrs);
if (rc < 0)
{
    fprintf(stderr, "Could not get attributes: %s, error %d\n", file, rc);
    exit(1);
}

printf("File name:%-20s\n", file);
printf("COS:%-20d\n", attrs.Attrs.COSId);
printf("EntryCount:%-20d\n", attrs.Attrs.EntryCount);
printf("Family:%-20d\n", attrs.Attrs.FamilyId);
```

The great thing about the hpss_FileGetAttributes() function is that it's relatively fast and in many cases its results could be returned from cache rather than being read from the database. Its downsides are mostly that there is a lot more information which can be retrieved and can be extremely useful - that information is the domain of the HPSS "extended attributes", not to be confused with the concept of linux extended attributes.

4.5.3.5 Get Extended Attributes

The HPSS extended attributes contain additional bitfile and storage server information about the location of the file data within the HPSS storage hierarchy and storage class. This information can be useful in making decisions in user applications because it's possible to tell, for example, whether the I/O request will require a tape mount.

```
int  rc;
char *file;
hpss_xfileattr_t attrs;
char buf[255];
int i, j, k;

login();

memset(&attrs, 0x0, sizeof(attrs));

file=argv[1];

rc = hpss_FileGetXAttributes(file, API_GET_STATS_FOR_ALL_LEVELS, 0, &attrs);
if (rc < 0)
{
    fprintf(stderr, "Could not get attributes: %s, error %d\n", file, rc);
    exit(1);
}

printf("File name:%-20s\n", file);
printf("COS:%-20d\n", attrs.Attrs.COSId);
printf("EntryCount:%-20d\n", attrs.Attrs.EntryCount);
printf("Family:%-20d\n", attrs.Attrs.FamilyId);

for(i=0;i<HPSS_MAX_STORAGE_LEVELS;i++)
{
   printf("Level %d Stripe Width: %d\n", i,                                    attrs.
      SCAttrib[i].StripeWidth);
   printf("Level %d Flags: %d\n", i, attrs.SCAttrib[i].Flags);
   printf("Level %d VVs: %d\n", i, attrs.SCAttrib[i].NumberOfVVs);

   for(j=0;j<attrs.SCAttrib[i].NumberOfVVs;j++)
   {
      printf("VV %d RelativePosition: %d\n", j,                                attrs.
      SCAttrib[i].VVAttrib[j].RelPosition);
      printf("VV %d BytesonVV: %s\n", j,                                       u64tostr_r(attrs.
      SCAttrib[i].VVAttrib[j].BytesOnVV, buf));
      if (attrs.SCAttrib[i].VVAttrib[j].PVList != NULL)
      {
         for(k=0; k < attrs.SCAttrib[i].VVAttrib[j].PVList->List.List_len; k++)
         {
            printf("PV Name %d: %s\n", k, attrs.SCAttrib[i].VVAttrib[j].
      PVList->List.List_val[k].Name);

         }
         free(attrs.SCAttrib[i].VVAttrib[j].PVList);
      }
   }
}
```

This snippet will present a few of the fields present in each level of detail within the extended attributes. The main concepts here are that there are three lists to go through, one for the levels of an HPSS hierarchy, one for the VV metadata within that hierarchy, and finally one for the list of PVs associated with those VVs. Using the extended attributes you can determine, for example, if the requested data might be on the second level of the hierarchy, but also that it's on disk, or maybe that the tape the data is on is shelved (not in the robotic library).

Of course getting all of this information can take some time. Most benchmarks place getting the extended attributes at anywhere from 100% to 400% slower than a simple file attributes retrieval. The performance can be improved if some scenarios can be removed and the requested information can be more targeted - the API allows for a specific level to be retrieved, for example. And of course, the PVList expects to be freed, so don't forget that.

4.5.3.6 Directory Listing

At a certain level, listing a directory just means getting the attributes for every namespace object in that directory. Within HPSS it is actually a fairly complex operation because an individual directory listing might have its ongoing database handle and results. This provides a consistent view of the chosen directory at a moment in time, and allows the request to be split across several requests from the Client API to the Name Server portion of the HPSS Core Server. Directories can also be extremely large and though directory results are normally sorted, in HPSS the default is not to sort them as that can require a lot of extra time which may be wasted if the user doesn't want or need a sorted listing. There is an option to retrieve a sorted listing for those who desire such output.

4.5.3.7 Short Listing

A short listing is relatively fast and simple to implement. The following program will list the objects contained in the provided directory name:

```
int  rc;
char *file;
hpss_dirent_t dirs;
int dirfd;
int i, j, k;
char* directory;

login();

memset(&dirs, 0x0, sizeof(dirs));

directory=argv[1];

dirfd = hpss_Opendir(directory);
if(dirfd < 0)
{
   fprintf(stderr, "Could not open directory %s, rc=%d\n", directory, dirfd);
   exit(1);
}

while(1)
{
   rc = hpss_Readdir(dirfd, &dirs);
   if (rc < 0)
   {
      fprintf(stderr, "Could not get attributes: %s, error %d\n", file, rc);
      (void) hpss_Closedir(dirfd);
      exit(1);
   }
   if(dirs.d_namelen == 0)
      break;

   printf("%s\n", dirs.d_name);
}

rc = hpss_Closedir(dirfd);
if(rc != 0)
{
   fprintf(stderr, "Could not close directory %d, rc=%d\n", dirfd, rc);
   exit(1);
}
```

Really not a lot to write home about here. Initially the directory is opened, and then we use the file descriptor returned by hpss_Opendir() to read the directory entries one by one. Once we see an entry with no name we'll stop reading them and close the directory - closing the directory will free up memory that would otherwise be left open until the open entry became stale as well as the file table entry associated with the open directory. Looking at the hpss_dirent_t structure, you may notice that there's not a lot of information here. Well, they don't call it a short listing for nothing. Next..

4.5.3.8 Long Listing

If you just read the previous section, "Short Listing", then this will look familiar to you. This is essentially the same program, but each entry in the directory has a hpss_GetAttrHandle() called on it to get its basic attributes.

```
int  rc;
char *file;
hpss_dirent_t dirs;
hpss_vattr_t attrs;
hpss_fileattr_t dir_attrs;
int dirfd;
int i, j, k;
char* directory;
ns_ObjHandle_t null_handle, cwd_handle;
char buf[255];

login();

memset(&dirs, 0x0, sizeof(dirs));
memset(&dir_attrs, 0x0, sizeof(dir_attrs));

directory=argv[1];

if(rc=hpss_FileGetAttributes(directory, &dir_attrs) < 0)
{
   fprintf(stderr, "Could not get attributes for %s, rc=%d\n", directory,
           rc);
}

cwd_handle = dir_attrs.ObjectHandle;

dirfd = hpss_Opendir(directory);
if(dirfd < 0)
{
   fprintf(stderr, "Could not open directory %s, rc=%d\n", directory, dirfd);
   exit(1);
}

printf("%20s %10s %10s %10s\n", "Name", "Size", "COS", "Account");
while(1)
{
   rc = hpss_Readdir(dirfd, &dirs);
   if (rc < 0)
   {
       fprintf(stderr, "Could not get directory error %d\n", rc);
       exit(1);
   }

   if(dirs.d_namelen == 0)
      break;

   rc = hpss_GetAttrHandle(&cwd_handle, dirs.d_name, NULL, NULL, &attrs);
   if(rc < 0)
   {
      fprintf(stderr, "Could not get attributes error %d\n", rc);
      exit(1);
   }

   printf("%20s %10s %10d %10d\n", dirs.d_name,
          u64tostr_r(attrs.va_size, buf), attrs.va_cos, attrs.va_account);
}

rc = hpss_Closedir(dirfd);
if(rc != 0)
{
   fprintf(stderr, "Could not close directory %d, rc=%d\n", dirfd, rc);
   exit(1);
}
```

This is a simple way to get file attributes. The entries come back one at a time and you simply get their attributes and print them out. This is one way to get file attributes, but another is to use the hpss_ReadAttrs() or hpss_Getdents() function calls. These will typically yield better performance than readdir because they can supply larger batches (hpss_Readdir() will read 128 entries at a time) and less overhead; for example, the directory entry retrieval and attribute retrieval operations occur within the same function rather than your application calling them for every entry.

4.5.4 File Input and Output

File I/O in HPSS at its most basic is familiar to anyone who has used the POSIX write and read functions before; however, there are really three modes of supported I/O in HPSS applications. One is the unbuffered APIs, another is the buffered APIs, and the third is PIO. The unbuffered APIs are simple to use and can provide good performance for small files where the overhead of setting up a complex I/O framework might take as long as just doing the I/O directly. However doing numerous small I/O within a small section of a file can be very expensive with the unbuffered APIs as each request actually writes data and generates a Core Server request.

The buffered APIs are built on top of the unbuffered APIs and to some extent hide the I/O latency by buffering write and read commands. The downside is that the buffered APIs can report success when writing data that was never actually written - the only way to know if the data made it to HPSS or not is by flushing the buffers periodically. The buffered APIs are styled after the C file stream APIs, although many options of those APIs are not currently supported by HPSS. Finally, PIO provides an abstraction of the basic protocol used to transfer data in HPSS rather than presenting a POSIX-like interface. This has the advantage of being faster than the POSIX interface as less requests must be made to the Core Server; PIO can stream hundreds of gigabytes of data between the client and the mover off a single Core Server request. The downside to PIO is that it's relatively expensive to set up and more complicated to implement. PIO also supports the PUSH protocol which is useful for WAN or high latency data movement.

It is important to remember that the underlying media will have a large impact on the performance characteristics of the I/O. Many customers use disk to tape hierarchies and so the tape is often no longer in use during I/O due to staging, but in the case where data is being written or read directly from tape the behavior of seek times or small I/O operations may have a noticeable delay due to tape positioning and startup times.

4.5.4.1 hpss_Write

```
int    hfd;
hpss_cos_hints_t  hints_in, hints_out;
hpss_cos_priorities_t  hints_pri;
char *hfile;
char *buf;
int rc;

login();

memset(&hints_in, 0x0, sizeof(hints_in));
memset(&hints_out, 0x0, sizeof(hints_out));
memset(&hints_pri, 0x0, sizeof(hints_pri));

hfile=argv[1];
buf=argv[2];

hfd = hpss_Open(hfile, O_RDWR, 0777,
                &hints_in, &hints_pri, &hints_out);
if (hfd < 0)
{
    fprintf(stderr, "Could not open: %s, error %d\n", hfile, hfd);
    exit(1);
}

rc = hpss_Write(hfd, buf, strlen(buf));
if(rc != strlen(buf))
{
   fprintf(stderr, "Could not write buf, rc=%d\n", rc);
   hpss_Close(hfd);
   exit(1);
}
fprintf(stderr, "Wrote %d bytes\n", rc);

hpss_Close(hfd);
```

hpss_Write() behaves similarly to the write system call. This snippet will write data passed in from the command line to an HPSS file specified at the command line. The file must already exist since the O_CREAT flag is not passed to hpss_Open(). hpss_Write() will return the number of bytes written. Since hpss_Write() is unbuffered the number of bytes written is the number of bytes that made it to the physical media and were committed by the database.

4.5.4.2 hpss_Read

```
int    hfd;
hpss_cos_hints_t  hints_in, hints_out;
hpss_cos_priorities_t  hints_pri;
char *hfile;
int rc;
char buf[255];

login();

memset(&hints_in, 0x0, sizeof(hints_in));
memset(&hints_out, 0x0, sizeof(hints_out));
memset(&hints_pri, 0x0, sizeof(hints_pri));

hfile=argv[1];

hfd = hpss_Open(hfile, O_RDWR, 0777,
                &hints_in, &hints_pri, &hints_out);
if (hfd < 0)
{
    fprintf(stderr, "Could not open: %s, error %d\n", hfile, hfd);
    exit(1);
}

rc = hpss_Read(hfd, buf, sizeof(buf));
if(rc <= 0)
{
   fprintf(stderr, "Could not read buf, rc=%d\n", rc);
   hpss_Close(hfd);
   exit(1);
}
fprintf(stderr, "Read %d bytes: %s\n", rc, buf);

hpss_Close(hfd);
```

hpss_Read() behaves similarly to the read system call. This will read up to 255 bytes of data from an HPSS file specified at the command line. hpss_Read() will return the number of bytes read into the supplied buffer.

4.5.4.3 PIO Concepts

PIO does not have a common analog, and requires at least two threads. The complexity lends PIO a great deal of flexibility at the price of being more difficult to use. A PIO application will typically outperform an application that uses the unbuffered or buffered API sets, especially for very large files or files which have a large number of segments.

Although PIO can be configured to use more than one client machine to transfer files, the following explanation will focus on the case where the only one client machine is used. The two roles within the PIO framework are the Coordinator and the Participant. The Coordinator role is given file I/O requests by the client application, forwards those requests to the HPSS Core Server, and manages the Participants which have registered with it. The Participant, which is a thread on the client machine, registers itself with the Coordinator's PIO group context and actually sends or receives the data, which is available to the client application in the form of a callback. The first step is to start the PIO coordinator thread and create the PIO group context. The options field can be used to specify whether to use PUSH or have PIO handle gaps. This is shown below:

```
hpss_pio_params_t params;
params.Operation        = HPSS_PIO_READ;
params.ClntStripeWidth = 1;
params.FileStripeWidth = 1;
params.BlockSize        = 4096;
params.IOTimeOutSecs    = 10;
params.Transport        = HPSS_PIO_TCPIP;
params.Options          = 0;

hpss_pio_grp_t sgrp;
int rc = hpss_PIOStart(&params, &sgrp);
```

Operation is either HPSS_PIO_READ or HPSS_PIO_WRITE for reads and writes to HPSS, respectively. ClntStripe-Width is the number of participant threads. Typically, you would want to create enough participants so that there is at least one participant for each stripe of the HPSS file, but this is not required. FileStripeWidth is the stripe width of the file on HPSS. This is typically determined by the COS. If you have a file on a COS with a top level storage class with a stripe width of four, then the FileStripeWidth of files created in that COS will be four. This information should probably be obtained when you open the HPSS file. The hints_out parameter of hpss_Open() will provide the file's stripe width. Alternatively, you can specify HPSS_PIO_STRIPE_UNKNOWN if you do not know and do not want to get this information elsewhere.

BlockSize is the buffer length that is passed into hpss_PIORegister() by each participant. Transport is either HPS-S_PIO_TCPIP or HPSS_PIO_MVR_SELECT. HPSS_PIO_MVR_SELECT is used for SAN3P, mainly.

There are several options which can be OR'd together and stored in Options. They are HPSS_PIO_PUSH, HP-SS_PIO_PORT_RANGE and HPSS_PIO_HANDLE_GAP. HPSS_PIO_PUSH is used to make the clients connect to the movers on writes to HPSS, which might be more efficient since it has fewer network connections. HPSS_-PIO_PORT_RANGE is used to specify that a range of ports can be used for the transfer on the client side. The range of ports is configured elsewhere. This has no effect when PUSH is used since the client is connecting to the mover. One thing to note is that a single file transfer using PIO can use several ports. Especially if the client is coming or going from a highly striped COS and thus has many participant threads. We have seen instances where over 30000 ports were used to transfer 10000 small files and ephemeral ports limits were reached. It is almost as easy to transfer many files with the one PIO setup as it is to transfer one file. This should be considered whenever many files are being transferred, especially if each transfer takes less than a minute. HPSS_PIO_HANDLE_GAP forces HPSS to handle the gaps which are encountered when transferring files. HPSS will zero-fill the buffer that is passed to the callback when it handles the gap. When you don't specify HPSS_PIO_HANDLE_GAP a call back will not be given for the gap. Instead, you must the hpss_pio_gapinfo_t pointer that was passed to hpss_PIOExecute(). The hpss_pio_gapinfo_t struct contains on any gaps which interrupted the hpss_PIOExecute(). You must use this information to handle the gap. This might include just seeking to the end of the gap and continuing with the read or write or writing the zeros over the gap. Here is an example of how you might handle the gaps:

```
      if(neqz64m(gap.Length) ||
   (neqz64m(gap.Offset)&&lt64m(currentOffset, pio->size)))
{
  U64_TO_ULL(gap.Length));
  inc64m(pio->total_moved, gap.Length);
  inc64m(io_bytes_moved, gap.Length);
  inc64m(currentOffset, gap.Length);
}
```

You can see that we did not manipulate our file in any way instead we only incremented our accounting variables. There is no need to zero fill buffers and then write those buffers although it is allowed. This section of code would be called after each time hpss_PIOExecute() returns without errors.

From there, the PIO group that was just created can be exported into a form useful by the participants. This is shown below:

```
rc = hpss_PIOExportGrp(sgrp, (void **)&group_buf, &group_buf_len);
```

Both group_buf and group_buf_len are output parameters - group_buf will be allocated by the hpss_PIOExportGrp() function and should be freed when the buffer is no longer needed. sgrp is the group that was output from hpss_PIO-Start() function. This provides a group context that can be used by participants to register with the coordinator. This group context which comes out of hpss_PIOExportGrp() is suitable for network transfer so the participants could even be on a separate machine.

At this point in the PIO process the coordinator is ready, but the participant(s) need to be created. These could be separate processes or threads. Once the participant has the group buffer (output from hpss_PIOExportGrp() as group_buf) it can then import the group buffer as shown below:

```
hpss_pio_grp_t sgrp;
int rc = hpss_PIOImportGrp(group_buf, group_buf_len, &sgrp);
```

This takes the network portable group_buf and decodes it back into a group context. The group context also contains the address of the coordinator. Now we're ready to register the participant with the coordinator. Once all of the participants are registered, the coordinator can begin issuing I/O. The coordinator knows how many participants there are from the ClntStripeWidth parameter supplied to hpss_PIOStart(). If you created fewer than that number of participants, then the coordinator thread will hang and timeout without issuing any I/O. Each call to hpss_PIO-Register() is done in a separate thread, so you would typically create a thread which allocates the buffer and does whatever other setup needs to be done and then calls hpss_PIORegister() once its ready to do I/O. The buffer length should be the same size as the BlockSize set in the params. If the buffer is larger than BlockSize then you will have wasted the difference, and if its smaller then hpss_PIORegister() will return an error.

```
rc = hpss_PIORegister(participant_index,
         NULL,
         buf,
         buflen,
         sgrp,
         pio_read_dump_buffer,
         NULL);
```

The participant is registered and requires the index of the participant (0-(N-1), where N is the client stripe width specified in the original hpss_PIOStart() parameters). The second parameter is an alternate data address to use for data connections - if NULL then the participant will listen as INADDR_ANY on a random port. The participant then supplies its data buffer, data buffer length, an I/O callback, I/O callback arguments, plus the group context it just imported.

Typically, you would create a struct that contains all the necessary information for each participant. This would include information such as local and HPSS file descriptors, buffer pointer, buffer len, etc. It might also be convenient to have a parent struct which contains transfer information such as file descriptors. This would allow you to transfer several files with only one call to hpss_PIOStart(). You also would not need to register your participants more than once. The example we use below has this type of design.

The I/O callback is important - it allows the application to access the data that is being read or written. Basically once PIO fills up a data buffer, it makes the finished buffer available to the application through the callback. Here's the full callback definition that was just registered by the hpss_PIORegister():

```
static int pio_read_dump_buffer(
   void             *arg,
   u_signed64       offset,
   unsigned int     *length,
   void             **buffer)
{
    pio_proc_t *proc = (pio_proc_t *)arg;
    pio_t *pio       = proc->parent;

    memcpy(proc->buf, *buffer, proc->buflen);
    unsigned char *readbuf = proc->buf;

    off_t noffset;
    CONVERT_U64_TO_LONGLONG(offset, noffset);
    if (lseek(pio->local_fd, noffset, SEEK_SET) == (off_t)-1) {
        fprintf(STDERR, "pio_read_dump_buffer(): lseek to %llu: %s",
                (long long)noffset, strerror(errno));
        return errno;
    }
    int total = 0;
    int left  = *length;
    unsigned char *bufPosition = readbuf;
    while (left) {
        int rc = write(pio->local_fd, bufPosition, left);
        if (rc < 0) {
            fprintf(STDERR, "pio_read_dump_buffer(): write: %s",
                        strerror(errno));
            return errno;
            }
        total       += rc;
        bufPosition += rc;
        left        -= rc;
        }

   return 0;
}
```

The first argument to the callback is the last argument that was given to hpss_PIORegister(). This void pointer could be different for each participant. Thus each participant can have its own buffer, or rather, should have its own buffer. You receive the void pointer and cast it as the proper struct. In this example, we cast arg as a pio_proc_t pointer, which contains a pointer to a base struct, pio_t, which contains the local and HPSS file descriptors that are being read to or from. This means you can change the place that the callback writes to just by replacing file descriptors in the pio_t struct.

The callback specifies a buffer and an offset; it's important to seek to the specified offset before using the data. This callback function copies the buffer containing the data read into the participant buffer and then writes it to the local file; a similar callback would be used for file writes where the callback would read a buffer from a source and supply it to PIO, and it would be registered in the same way.

Okay, so now we have a coordinator started, participants registered with appropriate callbacks. All that's left is to actually execute the I/O. This is done with the hpss_PIOExecute() function:

```
rc = hpss_PIOExecute(pio->hpss_fd, add64m(starting_offset,
                        current_offset), current_iosize,
                     sgrp, gap_ptr, &bytes_moved);
```

PIOExecute should be called in a loop until all of the I/O is received. In many cases, depending upon the storage class configuration, a single hpss_PIOExecute() will retrieve all of the data, however, that can never be counted on. Here you see the first mention of an open HPSS file. hpss_fd is an HPSS file descriptor that will be read / written. Following each hpss_PIOExecute() make sure to increment the current offset and decrement the current_iosize by the number of bytes moved. When a gap is encountered, the gap size should also be added to the offset.

This example has been largely about read, however performing PIO writes is essentially the same - the only differences are the operation passed in by the group parameters (HPSS_PIO_WRITE) and the callback (read from disk and load PIO buffer rather than load local buffer and write to disk).

4.5.5 HSM Functions

The HSM functions allow the application developer to control where the data resides in a class of service. When used in conjunction with the extended attributes, which provide the ability to see which levels data resides on, these functions can be used to ensure data is at the desired level. These functions can be used, for example, to force HPSS to immediately migrate a file to tape and then purge it from disk. It can also be used to pre-stage data that the application is aware will be needed soon from tape on to disk and force it to stay on disk.

It should be noted that these HSM functions, when improperly used, can result in inefficiency and wasted resources. For best results, migration and stage requests should be pooled and if possible ordered to reduce thrashing.

4.5.5.1 Migrate

The hpss_Migrate() function requires a user to have a special ACL with the Core Server. hpssmps is an authorized caller of the Core Server and so it may be used as an example. The reason for this is that migrations cause tape mounts and issuing migrations in an inefficient way could cause inefficient use of resources within the system vs. allowing the Migration Purge Server to manage migrations. Users who do not have the special ACL cannot issue migrations and their requests will be rejected.

The following code migrates a file and prints out the number of bytes moved. If a file has already been migrated from level 0, zero bytes will be moved. Keep in mind that hpss_Migrate() will not return until the file has been migrated or an error occurs, and so the program may take some time depending upon the size of the file and the characteristics of level 1 of the file's class of service.

```
int rc;
char buffer[255];
int i;
char* hfile;
int   hfd;
u_signed64 bytes = cast64m(0);

login();

hfile = argv[1];

hfd = hpss_Open(hfile, O_RDWR, 000, NULL, NULL, NULL);
if(hfd < 0)
{
   printf("Failed to open file %s, rc=%d\n", hfile, rc);
   return;
}

rc = hpss_Migrate(hfd, 0, 0, &bytes);

if(rc != 0)
{
   printf("Failed to Migrate %s, rc=%d\n", hfile, rc);
   return;
}
else
{
   printf("Migrated %s bytes\n", u64tostr_r(bytes, buffer));
}
```

4.5.5.2 Purge

The hpss_Purge() API is used to unlink data at one level of the hierarchy and allow that space to be reclaimed for use by other files. hpss_Purge() always requires that data exist at some other level - it will not allow data to be lost by purging. The most common usage is shown below, where a whole file is purged from the top level of the class of service. Unlike hpss_Migrate() which involves the movement of data and potentially tape mounts, etc, Purge is a metadata-only operation and so will be faster than Migrate for essentially all cases.

```
int rc;
char buffer[255];
int i;
char* hfile;
int   hfd;
u_signed64 bytes = cast64m(0);

login();

hfile = argv[1];

hfd = hpss_Open(hfile, O_RDWR, 000, NULL, NULL, NULL);
if(hfd < 0)
{
   printf("Failed to open file %s, rc=%d\n", hfile, hfd);
   return;
}

rc = hpss_Purge(hfd, cast64m(0), cast64m(0), 0, BFS_PURGE_ALL, &bytes);

if(rc != 0)
{
   printf("Failed to Purge %s, rc=%d\n", hfile, rc);
   return;
}
else
{
   printf("Purged %s bytes\n", u64tostr_r(bytes, buffer));
}
```

Like hpss_Migrate(), hpss_Purge() will report 0 bytes purged if no bytes existed at the level where the purge was requested.

4.5.5.3 Purge Lock

When scheduling data it can be important to make sure that the purge policies do not cause certain files to be purged off fast, top-level media. The following example opens a file, purge locks it, and then purge unlocks it. Remember that purge locks can have an expiration time set in the purge policy so it's important to keep in mind that a purge lock can be removed automatically, which can be a double-edged sword. The expiration time can be disabled if desired; consult the *HPSS Management Guide* for more information.

```
int rc;
char buffer[255];
int i;
char* hfile;
int   hfd = 0;
u_signed64 bytes = cast64m(0);

login();

hfile = argv[1];

hfd = hpss_Open(hfile, O_RDWR, 000, NULL, NULL, NULL);
if(hfd < 0)
{
   printf("Failed to open file %s, rc=%d\n", hfile, rc);
   return;
}

printf("Opened file %d\n", hfd);

rc = hpss_PurgeLock(hfd, PURGE_LOCK);
if(rc != 0)
{
   printf("Failed to purge lock %s, rc=%d\n", hfile, rc);
   return;
}
else
{
   printf("%s purge locked\n", hfile);
}

rc = hpss_PurgeLock(hfd, PURGE_UNLOCK);

if(rc != 0)
{
   printf("Failed to purge lock %s, rc=%d\n", hfile, rc);
   return;
}
else
{
   printf("%s purge unlocked\n", hfile);
}
```

Purge locks are only available for the top level of the hierarchy; if no data exists at the top level of the hierarchy an error is returned. This example immediately purge unlocks the file; in a real application you would likely wait for some condition to occur (such as the file being read by the waiting application) before doing the purge unlock.

4.5.5.4 Stage

As previously mentioned, when an application is aware that many stages may be upcoming, it is important to order the stage requests in such a way as to make the most efficient use of the tape drives involved. Unordered stages can lead to tapes being dismounted just before a stage request for a file on that tape is issued, or for a file at one end of a tape to be retrieved, followed by a file at the other end, followed by a file at the other end, etc. This can cause unnecessary delay and also wear and tear on the physical media.

Staging copies the file from a lower level up to a higher level of a storage hierarchy. The data still exists at the lower level, however. The following example opens a file and stages it. Keep in mind that if the COS is set up to stage on open you may need to include the O_NONBLOCK flag to keep the file from being staged.

```
int rc;
char buffer[255];
int i;
char* hfile;
int   hfd = 0;
u_signed64 bytes = cast64m(0);

login();

hfile = argv[1];

hfd = hpss_Open(hfile, O_RDWR | O_NONBLOCK, 000, NULL, NULL, NULL);
if(hfd < 0)
{
   printf("Failed to open file %s, rc=%d\n", hfile, rc);
   return;
}

printf("Opened file %d\n", hfd);

rc = hpss_Stage(hfd, 0, cast64m(0), 0, BFS_STAGE_ALL);
if(rc != 0)
{
   printf("Failed to stage %s, rc=%d\n", hfile, rc);
   return;
}
else
{
   printf("%s staged\n", hfile);
}
```

It's also important to keep in mind that tape is serial; it's possible that stages could be delayed significantly by some sort of long, ongoing I/O in progress on the same tape. That condition can be easily seen in the HPSS RTM screen where the stage request wait string will show it as blocked by another request.

4.5.6 User-defined Attributes (UDA)

The HPSS UDAs allow the arbitrary tagging of files with user-defined metadata in a hierarchical format. A full description of UDAs is out of the scope of this guide, however a few common guidelines are appropriate. -

1. If you plan to use an XML schema to manage the UDAs, it is important to define it early since any attributes stored before the schema is in place which violate the schema will keep that object from being updated once the schema is installed.
2. Batching UDAs operations is typically somewhat faster than issuing individual operations
3. Using the XQuery interface yields the best performance at the cost of additional design complexity.
4. Be very careful and test custom XQueries as it's easy to make mistakes and wind up with multiple entries of the same attribute name
5. Identify any attributes that should be searchable up front in your application design and create an XML index for each of them - searching attributes which do not have an index is very poor performing

4.5.6.1 Set UDAs

```
char *hfile;
int rc;
hpss_userattr_list_t inAttr;

login();

hfile=argv[1];

// set up UDA structure
inAttr.len = 1;
inAttr.Pair = malloc(inAttr.len * sizeof(hpss_userattr_t));
inAttr.Pair[0].Key = "/hpss/bestpractice/snippet";
inAttr.Pair[0].Value = "snip value";

// set the attribute
```

```
rc = hpss_UserAttrSetAttrs(hfile, &inAttr,NULL);
if(rc == 0)
   printf("Successfully set attribute\n");
else
   printf("Failed to set attribute, rc=%d\n", rc);

free(inAttr.Pair);
```

This snippet will create the attribute called "/hpss/bestpractice/snippet" if it does not exist and assign it the value of "snip value". If any value previously existed it is removed. The way these attribute names work is that they are hierarchical; they are referred to as "attribute paths" or "XPaths" because the UDA technology is based upon XQuery and XPath technology. The full UDA content for a file is stored within a single XML document within the database, and each update or retrieval references one or more paths within that XML document.

Large numbers of attributes can be set at once using the hpss_userattr_list_t structure, and those updates do happen within a single transaction.

4.5.6.2 Get UDA

Retrieving an attribute is even simpler than setting one. Just fill in the attribute pair with the key and the value will be retrieved. It's important to note that in the case where multiple attributes with the same name exist that the first attribute is retrieved by default. To specify a different attribute the one-based index must be provided (this is standard for XPath).

```
char *hfile;
int rc;
hpss_userattr_list_t inAttr;
int i;
int flag = UDA_API_VALUE;

login();

hfile=argv[1];


// set up UDA structure
inAttr.len = 1;
inAttr.Pair = malloc(inAttr.len * sizeof(hpss_userattr_t));
inAttr.Pair[0].Value = malloc(sizeof(char)*HPSS_XML_SIZE);
inAttr.Pair[0].Key = "/hpss/bestpractice/snippet";

rc = hpss_UserAttrGetAttrs(argv[1], &inAttr, flag, HPSS_XML_SIZE);

if(rc != 0)
{
   if(rc == -ENOENT)
      printf("No User-defined Attributes.\n");
   else
      printf("hpss_UserAttrGetAttrs returned %d\n", rc);
}
else
{
   for(i = 0; i < inAttr.len; i++) {
      char* xmlstr = hpss_ChompXMLHeader(inAttr.Pair[i].
      Value, NULL);
      printf("%s=\t\t%s\n", inAttr.Pair[i].Key, xmlstr);
      free(xmlstr);
   }
}

free(inAttr.Pair[0].Value);
free(inAttr.Pair);
```

This snippet will read the value of the attribute "/hpss/bestpractice/snippet" back and display it. Note that the hpss-_ChompXMLHeader() function is called on the string prior to it being displayed; each retrieved value which comes back always includes an XML header which is currently not used for anything, so we simply remove it.

Also, this example provides the output as a value, but it's also possible to retrieve the XML. This allows this function to be used to potentially return larger groups of values all under the same base attribute. For example, you could retrieve "/hpss/widget/a", "/hpss/widget/b", "/hpss/widget/c", and "/hpss/widget/d" at the same time by retrieving "/hpss/widget" in XML and then parsing that XML.

4.5.6.3 List UDA

You may not always know the attributes on a file, and simply want to see the full file contents. This is where the convenience function, hpss_UserAttrsListAttrs() comes in. hpss_UserAttrsListAttrs() is essentially a wrapper around an XML get of "/hpss" on a certain file. As described in the previous section, retrieving XML retrieves each of the children as well. List works by retrieving the entire XML document and then parsing it out into attribute key-value pairs.

```
char *hfile;
int rc;
hpss_userattr_list_t inAttr;
int i;
int flag = XML_ATTR;

login();

hfile=argv[1];


rc = hpss_UserAttrListAttrs(argv[1], &inAttr, flag, HPSS_XML_SIZE);

if(rc != 0)
{
   if(rc == -ENOENT)
      printf("No User-defined Attributes.\n");
   else
      printf("hpss_UserAttrListAttrs returned %d\n", rc);
}
else
{
   for(i = 0; i < inAttr.len; i++) {
      printf("%s=\t\t%s\n", inAttr.Pair[i].Key, inAttr.Pair[i].Value);
      free(inAttr.Pair[i].Value);
      free(inAttr.Pair[i].Key);
   }
}

free(inAttr.Pair);
```

The List API is surprisingly fast due to the fact that it only issues a single request to HPSS. The most common problem when doing a List is running into the ERANGE error. This is due to the application using a buffer size that is too small for the result set. In some cases an EDOM may be returned, which signifies that the Core Server buffer size limitation has been reached. In this case the limit may need to be increased or the attributes culled for space.

4.5.6.4 Delete UDA

Deleting attributes is nearly identical to setting them. The only difference is that no value should be provided as it will be ignored.

```
char *hfile;
int rc;
hpss_userattr_list_t inAttr;

login();

hfile=argv[1];

// set up UDA structure
inAttr.len = 1;
inAttr.Pair = malloc(inAttr.len * sizeof(hpss_userattr_t));
inAttr.Pair[0].Key = "/hpss/bestpractice/snippet";

// set the attribute
rc = hpss_UserAttrDeleteAttrs(hfile, &inAttr, NULL);
if(rc == 0)
   printf("Successfully deleted attribute\n");
else
   printf("Failed to delete attribute, rc=%d\n", rc);

free(inAttr.Pair);
```

There are a few notable differences between some delete functions in HPSS such as unlink and the User Defined Attributes delete. The first is that the UDA delete function will return success even if the attribute in question never existed. There are two reasons for this: technical and functional. The technical reason for this behavior is because that's how the XQuery delete behaves; an XQuery to delete a non-existing attribute is successful. The functional reason is that since multiple attributes can be deleted at once, even if some of the attributes no longer exist they can still be deleted without causing the entire operation to fail. The other way the UDA delete is different from say hpss_Unlink() is that it's essentially recursive with respect to child attributes. For example, in a file which might have /hpss/a/a, /hpss/a/b, and /hpss/a/c attributes, deleting /hpss/a will delete /hpss/a/a, /hpss/a/b, and /hpss/a/c. The path /text() can be appended on to the attribute in order to remove this recursive behavior and only delete a value associated with a particular attribute. For example /hpss/a/text() would return any value associated with /hpss/a, but not any of its child nodes such as /hpss/a/a.

4.5.6.5 Using Custom XQueries

The custom XQuery interface is an incredibly powerful tool that allows the application programmer to directly manipulate the XML document associated with the file. The best way to reduce the risk of a custom XQuery doing something unintended is through good application testing using a variety of user metadata.

First, an example of a custom XQuery:

```
char *xquery;
char *hfile;
int rc;

login();

hfile=argv[1];
xquery = "copy $new := $DOC modify(if ($new/hpss/test) then do replace value of $new/hpss/test with \"b\"
       else if($new/hpss) then do insert <test>{xs:string(\"b\")}</test> into $new/hpss else ()) return $new";

// set the attribute
rc = hpss_UserAttrXQueryUpdate(hfile, xquery, NULL);
if(rc == 0)
   printf("Successfully ran xquery update\n");
else
   printf("Failed to run xquery update, rc=%d\n", rc);
```

XQuery is a broad topic beyond the scope of this guide, however let's briefly go through the XQuery string above in order to give a hint of the complexity. So first, the XML document is copied into a sort of local variable ($new). Next the command is to modify this document. The logic must be split here because we may either be inserting a new node into the XML document or updating an existing node. One very important consideration is that every level within the XML document may need to be handled separately. For instance, in order to set /hpss/really/long/path to foo it may be necessary to have one conditional which updates path to foo, another which inserts

```
<path>
```

into /hpss/really/long with the value of foo, another which inserts

```
<long><path>
```

into /hpss/really with the value of foo, and so on. One thing the standard APIs never do that XQuery can easily do (and was hinted at in the previous paragraph) is insert and manipulate duplicate entries. For example, if the first condition was not there and the XQuery was simply..

```
"copy $new := $DOC modify(if($new/hpss) then do insert \
<test>{xs:string(\"b\")}</test> into $new/hpss else ()) return $new"
```

then every invocation of that XQuery would add another element called "test" with the value of "b" into /hpss. That's one of the major ways that the XQuery interface can be accidentally misused.

For more information on XQuery consult the DB2 documentation on the subject; not all operations are supported within DB2, and DB2 has a few special characteristics that need to be considered beyond the normal scope of XQuery.

4.5.6.6 Using Custom XQueries for Retrieval

The UDA data within a single file can be retrieved or manipulated prior to its retrieval using the hpss_UserAttrXQueryGet() API. This API supports the FLWOR syntax which is a very nice way to search through the XML tree. The FLWOR language is again out of scope for this guide, but like the other technologies it is well-documented by DB2 and in its own right. Here's an example which will print the number of key/value pairs that follow and up to the first ten keys and the values underneath the /hpss/a attribute in a comma and space-delimited format. For example, if two attributes exist under /hpss/a (/hpss/a/a=2 and /hpss/a/b=3) then the output would be: "2 a,2 b,3".

```
char *xquery;
int rc;
char buffer[HPSS_XML_SIZE];
int i;
int bufsize=10;
char* hfile;

login();

hfile = argv[1];
xquery = "let $j := for $v in $DOC/hpss/a/* return string-join((string(node-name($v)),data($v)),\",\")
       return insert-before(subsequence($j, 0, 10), 0, count($j))";

rc = hpss_UserAttrXQueryGet(hfile, xquery, buffer, sizeof(buffer));

if(rc != 0)
{
   printf("Failed to get xquery results, rc=%d\n", rc);
   return;
}
else
{
   printf("XQuery returned: %s\n", buffer);
}
```

Don't worry if the xquery doesn't make a lot of sense to you. Essentially FLWOR just allows the developer to iterate over a selection of nodes, perhaps apply some transformation, and return them. XQuery is an extremely flexible tool, and unlike the update function no updates occur so it's inherently less dangerous. XQuery is beyond the scope of this document, however there are plenty of resources on the Internet that describe XQuery and XPath.

4.5.6.7 Searching UDA

Searching UDAs is limited to users with special "trusted" status with the Core Server, such as root. Since the search capabilities could potentially include tens of millions of files tagged with UDA attributes it was important to restrict the harm that could be done by a single user or inefficient XQuery could do to the performance of the system.

That said, even administrators should consider their XQuery criteria carefully and follow the guidelines and advice of DB2 when crafting their XQuery searches. The two most important things in the author's experience are using wildcards sparingly, if ever, and having a lean XML index for your criteria. Now, an example:

```
char *xquery;
int subsys;
int rc;
object_id_list_t ObjectList;
char path[HPSS_MAX_PATH_NAME];
char idstr[24];
ns_ObjHandle_t fsroot;
u_signed64 id;
int i;
int bufsize=20;
hpss_cursor_id_t cursorid;

memset(&cursorid, 0x0, sizeof(cursorid));


login();

subsys=atoi(argv[1]);
xquery = "$DOC/hpss";

rc = hpss_UserAttrGetObjQuery(subsys, bufsize, &ObjectList,
                              xquery, &cursorid);
```

```
if(rc != 0)
  return;

printf("\n");
printf("Found %d objects:\n",  ObjectList.ObjectList.ObjectList_len);
printf("%24s\t\t%24s\n", "ID", "PATH");
printf("------------------------------------------------------------------\n");
for(i = 0; i < ObjectList.ObjectList.ObjectList_len; i++) {
   id = ObjectList.ObjectList.ObjectList_val[i];

   rc = hpss_GetPathObjId(id, subsys, &fsroot, path);
   if(rc != 0)
  {
      sprintf(path,"invalid path");
      free(ObjectList.ObjectList.ObjectList_val);
      return;
   }
   u64_to_decchar(id, idstr);
   printf("%24s\t\t%24s\n",idstr, path);
}

free(ObjectList.ObjectList.ObjectList_val);
```

This example will list every file which has an entry in the UDA table, and is probably an example of a search you wouldn't want to run very often on a system which makes heavy use of UDAs. The search criteria is essentially limited to XPaths plus attribute matching, which is outside of the scope of this document and widely documented on the Internet.

Essentially any XPath can be searched on, and can include wild cards or value matches. For example, to find all files that have the /hpss/color attribute set to green, the query is $DOC/hpss[color="green"]. XPaths can contain conjunctive operations such as ands and ors as well, although it should be noted that these can increase processing time substantially.

There are three types of search. ObjId returns the object id of the matching file and has been by far the most useful in most user's experience. XML returns the matching XML, but not any file information. XMLObjId returns the object id and matching XML, and can also be used in a double constraint mode that works like "Find Object and XML matching Query 1 where Object also matches Query 2". The usage of these functions is essentially identical to the ObjId case but instead of Object Ids the search returns XML or XML and Object Ids.

The search APIs return data in sets, and in many or even most cases the result set may be larger than the set size specified. In this case an alternative search function may be called to retrieve additional results from the same query. Here's an example of that:

```
while(!done)
{
   rc = hpss_UserAttrReadObjs(Subsys, &objs, &cursorid);
   if(rc != HPSS_E_NOERROR)
   {
      (void)hpss_UserAttrCloseCursor(Subsys, &cursorid);
      break;
   }
   else
   {
      // Let's convert these new object ids to path names and print them
      for(i = 0; i < objs.ObjectList.ObjectList_len; i++)
      {
         rc = hpss_GetPathObjId(objs.ObjectList.ObjectList_val[i], Subsys, &fsroot, path);
         if(rc == HPSS_E_NOERROR)
            printf("%d) %s\n", idx++, path);
         else
            printf("%d) error parsing path name, obj=%u.%u, rc=%d", idx++,
                   high32m(objs.ObjectList.ObjectList_val[i]),
                   low32m(objs.ObjectList.ObjectList_val[i]), rc);
      }

      free(objs.ObjectList.ObjectList_val);
   }

}
```

The previous snippet will continue retrieving data from the original search using the cursor structure until an error occurs. When no more results are available the ReadObjs function returns a ESRCH. In this way millions of search result entries can be retrieved from HPSS, although in that case it may be prudent to use a buffer size larger than 20 entries.

4.5.7 Using UDAs with VFS

VFS uses the UDA namespace /hpss/fs/user. By default any attributes which match that pattern will be accessible via VFS using standard extended attributes tools. For example, setting an attribute "/hpss/fs/user.test" would be visible in VFS as "user.test". Attributes set in VFS will be updated in the same way. Any attributes which are not under the /hpss/fs/user namespace will not be visible via HPSS VFS, with a few exceptions described below.

In order to make use of the standard HPSS checksum attributes, VFS maps the HPSS checksum attribute names to names which are compatible with VFS; it maps them under the /hpss/fs/user XPath. Applications or users setting or getting these checksum attributes via VFS must use these translated attribute names rather than the standard HPSS checksum paths. The following table describes the mapping:

HPSS Checksum Attribute	VFS Mapped Checksum Attribute
/hpss/user/cksum/checksum	/hpss/fs/user.hash.checksum
/hpss/user/cksum/algorithm	/hpss/fs/user.hash.algorithm
/hpss/user/cksum/state	/hpss/fs/user.hash.state
/hpss/user/cksum/lastupdate	/hpss/fs/user.hash.lastupdate
/hpss/user/cksum/errors	/hpss/fs/user.hash.errors
/hpss/user/cksum/filesize	/hpss/fs/user.hash.filesize
/hpss/user/cksum/app	/hpss/fs/user.hash.app

4.5.8 HPSS to POSIX and Back Again

There are several structures returned by HPSS which do not provide what would be considered standard POSIX information. Luckily, there are convenience functions which take this information and put it into a POSIX format. While the information doesn't always match one-to-one with POSIX, it is as close a translation as possible.

4.5.8.1 HPSS Mode to POSIX Mode

```
int  rc;
char *file;
hpss_fileattr_t attrs;
mode_t mode;

login();

memset(&attrs, 0x0, sizeof(attrs));

file=argv[1];

rc = hpss_FileGetAttributes(file, &attrs);
if (rc < 0)
{
    fprintf(stderr, "Could not get attributes: %s, error %d\n", file, rc);
    exit(1);
}

API_ConvertModeToPosixMode(&attrs.Attrs, &mode);
```

This will translate the hpss_Attrs_t Type, ModePerms, UserPerms, GroupsPerms, and OtherPerms fields into a mode_t, with its bit fields set with appropriate values based upon the machine where the POSIX conversion function ran. The attributes returned by this conversion are OS-specific and machine-specific, and care should be taken when using these mode bits in a mixed environment. There's also a reciprocal function which takes a POSIX mode and sets the appropriate fields in an HPSS Attr structure.

4.5.8.2 HPSS Time to POSIX Time

HPSS has timestamps for creation time, modified time, write time, and read time while POSIX has access time, modify time, and change time. The API_ConvertTimeToPosixTime() function converts the HPSS timestamps as nearly as possible to the POSIX timestamps.

```
int  rc;
char *file;
hpss_fileattr_t attrs;
timestamp_sec_t atime;
timestamp_sec_t mtime;
timestamp_sec_t ctime;


login();

memset(&attrs, 0x0, sizeof(attrs));

file=argv[1];

rc = hpss_FileGetAttributes(file, &attrs);
if (rc < 0)
{
    fprintf(stderr, "Could not get attributes: %s, error %d\n", file, rc);
    exit(1);
}

API_ConvertTimeToPosixTime(&attrs.Attrs, &atime, &mtime, &ctime);

printf("a=%d, m=%d, c=%d\n", atime, mtime, ctime);
```

You can expect to see a few anomalies - HPSS does not strictly follow POSIX guidelines for updating timestamps. Some timestamps may be updated in a lazy fashion, for example. In addition, some timestamps could be updated based upon system activities such as migrations or purges.

4.5.9 More On UDAs

4.5.9.1 Guidelines, Best Practices, and Standard Attributes

There are several guidelines that should be followed when using UDAs in your application. The first is that, in order to maintain namespace uniqueness, the application should place all of its attributes under a common XPath name (perhaps some form of the application name). For example, the following is good usage which allows for uniqueness:

UDA Description	XPath
Date	/hpss/myapp/date
User	/hpss/myapp/user
Id	/hpss/myapp/id

Here's an example of some poor attribute usage:

UDA Description	XPath
Date	/hpss/data
User	/hpss/user
Id	/hpss/id

Without including an application identifier to put these application attributes inside, the attributes reside on the top level. If this is done by multiple applications, then there's a good chance that common attribute names could be overwritten by poor behaving applications. Using an application identifier safeguards these fairly common attribute names from accidental overwrites by other applications.

Another guideline is that, since many applications do checksumming, there a common location and set of attributes has been defined where checksums may be stored. Storing checksums in this way will allow the applications to be compatible with each other and HPSS tools such as **hpsssum**. These attributes and their meanings are listed below:

UDA Description	VFS Path	Comments
Checksum Hash	/hpss/user/cksum/checksum	The checksum hash in hex format. All lower case is preferred, but it should be case insensitive.
Algorithm Name	/hpss/user/cksum/algorithm	See the **hpsssum** man page for a list of checksum names that HPSS supports. If an application or tool does not support the specified checksum type it should error out. Algorithm names should be case insensitive.
Checksum State	/hpss/user/cksum/state	Legal values: "Valid" - No errors detected validating the checksum "Error" - The last validation attempt was unsuccessful due to an error. "Invalid" - Validation failed; the generated checksum during a previous read did not match the saved checksum. Files which are in "Error" or "Valid" should open normally, files which are "Invalid" should fail to open with EILSEQ or allow the user to determine whether to continue or not. Checksum state should be case insensitive.
Last time the checksum was updated	/hpss/user/cksum/lastupdate	This is the time of the last checksum hash attribute update in seconds since Epoch.
Number of errors reading the file	/hpss/user/cksum/errors	Informational only; may be reset to 0 following a successful read and verification of the file against the stored checksum.
Size of the check-summed file	/hpss/user/cksum/filesize	This is the size, in bytes, of the checksummed file.
Application Name	/hpss/user/cksum/app	This is the name of the application which last updated the checksum hash. Using an unsupported algorithm name will result in **hpsssum** being unable to verify the file. **hpsssum** can be used to identify files which are in the 'error' or 'invalid' state, although it is recommended that XML indexes be created if this feature is intended to be used in a production scenario. (See the xmladdindex man page).

4.5.10 Useful References

1. `http://www.ibm.com/developerworks/data/library/techarticle/dm-0604saracco/`
   - Using XQuery
2. `http://www.ibm.com/developerworks/data/library/techarticle/dm-0710nicola/`
   - Using XQuery Update
3. `https://scholar.google.com/scholar?hl=en&as_sdt=0%2C44&q=Using+xml+and+xquery+for+da`
   `G=` - Using xml and xquery for data management in hpss

Chapter 5

Developer Acknowledgements

HPSS is a product of a government-industry collaboration. The project approach is based on the premise that no single company, government laboratory, or research organization has the ability to confront all of the system-level issues that must be resolved for significant advancement in high-performance storage system technology.

HPSS development was performed jointly by IBM Worldwide Government Industry, Lawrence Berkeley National Laboratory, Lawrence Livermore National Laboratory, Los Alamos National Laboratory, NASA Langley Research Center, Oak Ridge National Laboratory, and Sandia National Laboratories.

We would like to acknowledge Argonne National Laboratory, the National Center for Atmospheric Research, and Pacific Northwest Laboratory for their help with initial requirements reviews.

We also wish to acknowledge Cornell Information Technologies of Cornell University for providing assistance with naming service and transaction management evaluations and for joint developments of the Name Service.

In addition, we wish to acknowledge the many discussions, design ideas, implementation and operation experiences we have shared with colleagues at the National Storage Laboratory, the IEEE Mass Storage Systems and Technology Technical Committee, the IEEE Storage System Standards Working Group, and the storage community at large.

We also wish to acknowledge the Cornell Theory Center and the Maui High Performance Computer Center for providing a test bed for the initial HPSS release. We also wish to acknowledge Gleicher Enterprises, LLC for the development of the HSI, HTAR and Transfer Agent client applications.

Finally, we wish to acknowledge CEA-DAM (Commissariat a l'Energie Atomique - Centre d'Etudes de Bruyeres-le--Chatel) for providing assistance with development of NFS V3 protocol support.

Chapter 6

Further Reading

1. *3580 Ultrium Tape Drive Setup, Operator and Service Guide* GA32-0415-00
2. *3584 UltraScalable Tape Library Planning and Operator Guide* GA32-0408-01
3. *3584 UltraScalable Tape Library SCSI Reference* WB1108-00
4. *HPSS Error Messages Reference Manual*, current release.
5. *HPSS Programmer's Reference* , current release.
6. *HPSS Programmer's Reference* - I/O Supplement , current release.
7. *HPSS User's Guide*, current release.
8. *IBM SCSI Device Drivers: Installation and User's Guide*, GC35-0154-01
9. *IBM Ultrium Device Drivers Installation and User's Guide* GA32-0430-00.1
10. Parallel and ESCON Channel Tape Attachment/6000 Installation and User's Guide, GA32-0311-02
11. POSIX 1003.1-1990 Tar Standard
12. Reference Guide AMU, Order no. DOC E00 005
13. STK Automated Cartridge System Library Software (ACSLS) System Administrator's Guide, PN 16716
14. STK Automated Cartridge System Library Software Programmer's Guide, PN 16718
15. J. Steiner, C. Neuman, and J. Schiller, "Kerberos: An Authentication Service for Open Network Systems," USENIX 1988 Winter Conference Proceedings (1988).
16. R.W. Watson and R.A. Coyne, "The Parallel I/O Architecture of the High-Performance Storage System (HPS-S)," from the 1995 IEEE MSS Symposium, courtesy of the IEEE Computer Society Press.
17. T.W. Tyler and D.S. Fisher, "Using Distributed OLTP Technology in a High-Performance Storage System," from the 1995 IEEE MSS Symposium, courtesy of the IEEE Computer Society Press.
18. J.K. Deutsch and M.R. Gary, "Physical Volume Library Deadlock Avoidance in a Striped Media Environment," from the 1995 IEEE MSS Symposium, courtesy of the IEEE Computer Society Press.
19. R. Grossman, X. Qin, W. Xu, H. Hulen, and T. Tyler, "An Architecture for a Scalable, High-Performance Digital Library," from the 1995 IEEE MSS Symposium, courtesy of the IEEE Computer Society Press.
20. S. Louis and R.D. Burris, "Management Issues for High-Performance Storage Systems," from the 1995 IEEE MSS Symposium, courtesy of the IEEE Computer Society Press.
21. D. Fisher, J. Sobolewski, and T. Tyler, "Distributed Metadata Management in the High Performance Storage System," from the 1st IEEE Metadata Conference, April 16-18, 1996.

Chapter 7

Glossary of Terms and Acronyms

ACI Automatic Media Library Client Interface

ACL Access Control List

ACSLS Automated Cartridge System Library Software (Science Technology Corporation)

ADIC Advanced Digital Information Corporation

accounting The process of tracking system usage per user, possibly for the purposes of charging for that usage. Also, a log record type used to log accounting information.

AIX Advanced Interactive Executive. An operating system provided on many IBM machines.

alarm A log record type used to report situations that require administrator investigation or intervention.

AML Automated Media Library. A tape robot.

AMS Archive Management Unit

ANSI American National Standards Institute

API Application Program Interface

archive One or more interconnected storage systems of the same architecture.

attribute When referring to a managed object, an attribute is one discrete piece of information, or set of related information, within that object.

attribute change When referring to a managed object, an attribute change is the modification of an object attribute. This event may result in a notification being sent to SSM, if SSM is currently registered for that attribute.

audit (security) An operation that produces lists of HPSS log messages whose record type is SECURITY. A security audit is used to provide a trail of security-relevant activity in HPSS.

bar code An array of rectangular bars and spaces in a predetermined pattern which represent alphanumeric information in a machine readable format (e.g., UPC symbol)

BFS HPSS Bitfile Service.

bitfile A file stored in HPSS, represented as a logical string of bits unrestricted in size or internal structure. HPSS imposes a size limitation in 8-bit bytes based upon the maximum size in bytes that can be represented by a 64-bit unsigned integer.

bitfile segment An internal metadata structure, not normally visible, used by the Core Server to map contiguous pieces of a bitfile to underlying storage.

Bitfile Service Portion of the HPSS Core Server that provides a logical abstraction of bitfiles to its clients.

bytes between tape marks The number of data bytes that are written to a tape virtual volume before a tape mark is required on the physical media.

cartridge A physical media container, such as a tape reel or cassette, capable of being mounted on and dismounted from a drive. A fixed disk is technically considered to be a cartridge because it meets this definition and can be logically mounted and dismounted.

central log The main repository of logged messages from all HPSS servers. Usually /var/hpss/log/HPSS.log.

class A type definition in Java. It defines a template on which objects with similar characteristics can be built, and includes variables and methods specific to the class.

Class of Service A set of storage system characteristics used to group bitfiles with similar logical characteristics and performance requirements together. A Class of Service is supported by an underlying hierarchy of storage classes.

cluster The unit of storage space allocation on HPSS disks. The smallest amount of disk space that can be allocated from a virtual volume is a cluster. The size of the cluster on any given disk volume is determined by the size of the smallest storage segment that will be allocated on the volume, and other factors.

configuration The process of initializing or modifying various parameters affecting the behavior of an HPSS server or infrastructure service.

COS Class of Service

Core Server An HPSS server which manages the namespace and storage for an HPSS system. The Core Server manages the Name Space in which files are defined, the attributes of the files, and the storage media on which the files are stored. The Core Server is the central server of an HPSS system. Each storage sub-system uses exactly one Core Server.

daemon A UNIX program that runs continuously in the background.

DB2 A relational database system, a product of IBM Corporation, used by HPSS to store and manage HPSS system metadata.

debug A log record type used to report internal events that can be helpful in troubleshooting the system.

delog The process of extraction, formatting, and outputting HPSS central log records. This process is obsolete in 7.4 and later versions of HPSS. The central log is now recorded as flat text.

deregistration The process of disabling notification to SSM for a particular attribute change.

descriptive name A human-readable name for an HPSS server.

device A physical piece of hardware, usually associated with a drive, that is capable of reading or writing data.

directory An HPSS object that can contain files, symbolic links, hard links, and other directories.

dismount An operation in which a cartridge is either physically or logically removed from a device, rendering it unreadable and unwritable. In the case of tape cartridges, a dismount operation is a physical operation. In the case of a fixed disk unit, a dismount is a logical operation.

DNS Domain Name Service

DOE Department of Energy

drive A physical piece of hardware capable of reading and/or writing mounted cartridges. The terms device and drive are often used interchangeably.

event A log record type used to report informational messages (e.g., subsystem starting, subsystem terminating).

export An operation in which a cartridge and its associated storage space are removed from the HPSS system Physical Volume Library. It may or may not include an eject, which is the removal of the cartridge from its Physical Volume Repository.

file An object than can be written to, read from, or both, with attributes including access permissions and type, as defined by POSIX (P1003.1-1990). HPSS supports only regular files.

file family An attribute of an HPSS file that is used to group a set of files on a common set of tape virtual volumes.

fileset A collection of related files that are organized into a single easily managed unit. A fileset is a disjoint directory tree that can be mounted in some other directory tree to make it accessible to users.

fileset ID A 64-bit number that uniquely identifies a fileset.

fileset name A name that uniquely identifies a fileset.

file system ID A 32-bit number that uniquely identifies an aggregate.

FTP File Transfer Protocol

Gatekeeper An HPSS server that provides two main services: the ability to schedule the use of HPSS resources referred to as the Gatekeeping Service, and the ability to validate user accounts referred to as the Account Validation Service.

Gatekeeping Service A registered interface in the Gatekeeper that provides a site the mechanism to create local policy on how to throttle or deny create, open and stage requests and which of these request types to monitor.

Gatekeeping Site Interface The APIs of the gatekeeping site policy code.

Gatekeeping Site Policy The gatekeeping shared library code written by the site to monitor and throttle create, open, and/or stage requests.

GB Gigabyte (230)

GECOS The comment field in a UNIX password entry that can contain general information about a user, such as office or phone number.

GID Group Identifier

GK Gatekeeper

GSS Generic Security Service

GUI Graphical User Interface

HA High Availability

HACMP High Availability Clustered Multi-Processing - A software package used to implement high availability systems.

halt A forced shutdown of an HPSS server.

hierarchy See Storage Hierarchy.

HPSS High Performance Storage System

HPSS-only fileset An HPSS fileset that is not linked to an external filesystem.

IBM International Business Machines Corporation

ID Identifier

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

import An operation in which a cartridge and its associated storage space are made available to the HPSS system. An import requires that the cartridge has been physically introduced into a Physical Volume Repository (injected). Importing the cartridge makes it known to the Physical Volume Library.

I/O Input/Output

IOD/IOR Input/Output Descriptor / Input/Output Reply. Structures used to send control information about data movement requests in HPSS and about the success or failure of the requests.

IP Internet Protocol

IRIX SGI's implementation of UNIX

junction A mount point for an HPSS fileset.

KB Kilobyte (210)

KDC Key Distribution Center

LAN Local Area Network

LANL Los Alamos National Laboratory

LARC Langley Research Center

latency For tape media, the average time in seconds between the start of a read or write request and the time when the drive actually begins reading or writing the tape.

LDAP Lightweight Directory Access Protocol

LLNL Lawrence Livermore National Laboratory

local log An optional log on a remote system, for example on a Mover node under /var/hpss/log/HPSS.log. Remote system logs should be forwarded to the central logging system.

Location Server An HPSS server that is used to help clients locate the appropriate Core Server and/or other HPSS server to use for a particular request.

log record A message generated by an HPSS application and handled and recorded by the HPSS logging subsystem.

log record type A log record may be of type alarm, event, status, debug, request, security, trace, or accounting.

LRU Least Recently Used

LS Location Server

LTO Linear Tape-Open. A half-inch open tape technology developed by IBM, HP and Seagate.

MAC Mandatory Access Control

managed object A programming data structure that represents an HPSS system resource. The resource can be monitored and controlled by operations on the managed object. Managed objects in HPSS are used to represent servers, drives, storage media, jobs, and other resources.

MB Megabyte (220)

metadata Control information about the data stored under HPSS, such as location, access times, permissions, and storage policies. Most HPSS metadata is stored in a DB2 relational database.

method A Java function or subroutine

migrate To copy file data from a level in the file's hierarchy onto the next lower level in the hierarchy.

Migration/Purge Server An HPSS server responsible for supervising the placement of data in the storage hierarchies based upon site-defined migration and purge policies.

MM Metadata Manager. A software library that provides a programming API to interface HPSS servers with the DB2 programming environment.

mount An operation in which a cartridge is either physically or logically made readable and/or writable on a drive. In the case of tape cartridges, a mount operation is a physical operation. In the case of a fixed disk unit, a mount is a logical operation.

mount point A place where a fileset is mounted in HPSS.

Mover An HPSS server that provides control of storage devices and data transfers within HPSS.

MPS Migration/Purge Server

MRA Media Recovery Archive

MSSRM Mass Storage System Reference Model

MVR Mover

NASA National Aeronautics and Space Administration

Name Service The portion of the Core Server that providesa mapping between names and machine oriented identifiers. In addition, the Name Service performs access verification and provides the Portable Operating System Interface (POSIX). name space The set of name-object pairs managed by the HPSS Core Server.

NERSC National Energy Research Supercomputer Center

NLS National Language Support

notification A notice from one server to another about a noteworthy occurrence. HPSS notifications include notices sent from other servers to SSM of changes in managed object attributes, changes in tape mount information, and log messages of type alarm, event, or status.

NS HPSS Name Service

NSL National Storage Laboratory

object See Managed Object

ORNL Oak Ridge National Laboratory

OSF Open Software Foundation

OS/2 Operating System (multi-tasking, single user) used on the AMU controller PC

PB Petabyte (250)

PFTP Parallel File Transfer Protocol

physical volume An HPSS object managed jointly by the Core Server and the Physical Volume Library that represents the portion of a virtual volume. A virtual volume may be composed of one or more physical volumes, but a physical volume may contain data from no more than one virtual volume.

Physical Volume Library An HPSS server that manages mounts and dismounts of HPSS physical volumes.

Physical Volume Repository An HPSS server that manages the robotic agent responsible for mounting and dismounting cartridges or interfaces with the human agent responsible for mounting and dismounting cartridges.

PIO Parallel I/O

PIOFS Parallel I/O File System

POSIX Portable Operating System Interface (for computer environments)

purge Deletion of file data from a level in the file's hierarchy after the data has been duplicated at lower levels in the hierarchy and is no longer needed at the deletion level.

purge lock A lock applied to a bitfile which prohibits the bitfile from being purged.

PV Physical Volume

PVL Physical Volume Library

PVR Physical Volume Repository

RAID Redundant Array of Independent Disks

RAIT Redundant Array of Independent Tapes

RAM Random Access Memory

reclaim The act of making previously written but now empty tape virtual volumes available for reuse. Reclaimed tape virtual volumes are assigned a new Virtual Volume ID, but retain the rest of their previous characteristics. Reclaim is also the name of the utility program that performs this task.

registration The process by which SSM requests notification of changes to specified attributes of a managed object.

reinitialization An HPSS SSM administrative operation that directs an HPSS server to reread its latest configuration information, and to change its operating parameters to match that configuration, without going through a server shutdown and restart.

repack The act of moving data from a virtual volume onto another virtual volume with the same characteristics with the intention of removing all data from the source virtual volume. Repack is also the name of the utility program that performs this task.

request A log record type used to report some action being performed by an HPSS server on behalf of a client.

RISC Reduced Instruction Set Computer/Cycles

RMS Removable Media Service

RPC Remote Procedure Call

SCSI Small Computer Systems Interface

security A log record type used to report security related events (e.g., authorization failures).

SGI Silicon Graphics

shelf tape A cartridge which has been physically removed from a tape library but whose file metadata still resides in HPSS.

shutdown An HPSS SSM administrative operation that causes a server to stop its execution gracefully.

sink The set of destinations to which data is sent during a data transfer (e.g., disk devices, memory buffers, network addresses).

SMC SCSI Medium Changer

SMIT System Management Interface Tool

SNL Sandia National Laboratories

SOID Storage Object ID. An internal HPSS storage object identifier that uniquely identifies a storage resource. The SOID contains an unique identifier for the object, and an unique identifier for the server that manages the object.

source The set of origins from which data is received during a data transfer (e.g., disk devices, memory buffers, network addresses).

SP Scalable Processor

SS HPSS Storage Service

SSA Serial Storage Architecture

SSM Storage System Management

SSM session The environment in which an SSM user interacts with the SSM System Manager to monitor and control HPSS. This environment may be the graphical user interface provided by the hpssgui program, or the command line user interface provided by the hpssadm program.

stage To copy file data from a level in the file's hierarchy onto the top level in the hierarchy.

start-up An HPSS SSM administrative operation that causes a server to begin execution.

status A log record type used to report progress for long running operations.

STK Storage Technology Corporation

storage class An HPSS object used to group storage media together to provide storage for HPSS data with specific characteristics. The characteristics are both physical and logical.

storage hierarchy An ordered collection of storage classes. The hierarchy consists of a fixed number of storage levels numbered from level 1 to the number of levels in the hierarchy, with the maximum level being limited to 5 by HPSS. Each level is associated with a specific storage class. Migration and stage commands result in data being copied between different storage levels in the hierarchy. Each Class of Service has an associated hierarchy.

storage level The relative position of a single storage class in a storage hierarchy. For example, if a storage class is at the top of a hierarchy, the storage level is 1.

storage map An HPSS object managed by the Core Server to keep track of allocated storage space.

storage segment An HPSS object managed by the Core Server to provide abstract storage for a bitfile or parts of a bitfile.

Storage Service The portion of the Core Server which provides control over a hierarchy of virtual and physical storage resources.

storage subsystem A portion of the HPSS namespace that is managed by an independent Core Server and (optionally) Migration/Purge Server.

Storage System Management An HPSS component that provides monitoring and control of HPSS via a windowed operator interface or command line interface.

stripe length The number of bytes that must be written to span all the physical storage media (physical volumes) that are grouped together to form the logical storage media (virtual volume). The stripe length equals the virtual volume block size multiplied by the number of physical volumes in the stripe group (i.e., stripe width).

stripe length The number of bytes that must be written to span all the physical storage media (physical volumes) that are grouped together to form the logical storage media (virtual volume). The stripe length equals the virtual volume block size multiplied by the number of physical volumes in the stripe group (i.e., stripe width).

stripe width The number of physical volumes grouped together to represent a virtual volume.

System Manager The Storage System Management (SSM) server. It communicates with all other HPSS components requiring monitoring or control. It also communicates with the SSM graphical user interface (hpssgui) and command line interface (hpssadm).

TB Terabyte (240)

TCP/IP Transmission Control Protocol/Internet Protocol

trace A log record type used to record procedure entry/exit events during HPSS server software operation.

transaction A programming construct that enables multiple data operations to possess the following properties: All operations commit or abort/roll-back together such that they form a single unit of work. All data modified as part of the same transaction are guaranteed to maintain a consistent state whether the transaction is aborted or committed. Data modified from one transaction are isolated from other transactions until the transaction is either committed or aborted. Once the transaction commits, all changes to data are guaranteed to be permanent.

TTY Teletypewriter

UDA User-defined Attribute

UDP User Datagram Protocol

UID User Identifier

UPC Universal Product Code

UUID Universal Unique Identifier

virtual volume An HPSS object managed by the Core Server that is used to represent logical media. A virtual volume is made up of a group of physical storage media (a stripe group of physical volumes).

virtual volume block size The size of the block of data bytes that is written to each physical volume of a striped virtual volume before switching to the next physical volume.

VV Virtual Volume

XDSM The Open Group's Data Storage Management standard. It defines APIs that use events to notify Data Management applications about operations on files.

XFS A file system created by SGI available as open source for the Linux operating system.

XML Extensible Markup Language

Chapter 8

Deprecated List

Member __hpss_net_maskaddr32 (const hpss_sockaddr_t ∗addr, signed32 mask)

7.4.1

Member api_config::DescName [HPSS_MAX_DESC_NAME]

Remove in next major release.

Member api_config::DMAPWriteUpdates

This field has been unused since HPSS 6.

Member hpss_FilesetCreate (const hpss_srvr_id_t ∗CoreServerID, uint32_t CreateOptions, ns_FilesetAttrBits_t FilesetAttrBits, const ns_FilesetAttrs_t ∗FilesetAttrs, hpss_AttrBits_t ObjectAttrBits, const hpss_Attrs_t ∗ObjectAttrs, ns_FilesetAttrBits_t RetFilesetAttrBits, hpss_AttrBits_t RetObjectAttrBits, ns_FilesetAttrs_t ∗RetFilesetAttrs, hpss_Attrs_t ∗RetObjectAttrs, ns_ObjHandle_t ∗FilesetHandle)

The *CreateOptions* parameter of this function will be removed in a future version of HPSS.

Member hpss_Getdents (const ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_t ∗Ucred, uint32_t BufferSize, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

This function will be removed in a future version of HPSS. Use hpss_ReaddirPlus().

Member hpss_GetdentsAttrs (ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_t ∗Ucred, uint32_t BufferSize, uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

This function will be removed in a future version of HPSS. Use hpss_ReadAttrsPlus().

Member hpss_PIOEnd (hpss_pio_grp_t StripeGroup)

Deprecated in HPSS 7.5; use hpss_PIOFinalize() instead.

Member hpss_ReadAttrs (const int Dirdes, const uint64_t OffsetIn, const uint32_t BufferSize, const uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

This function will be removed in a future version of HPSS. Use hpss_ReadAttrsPlus().

Member hpss_ReadAttrsHandle (const ns_ObjHandle_t ∗ObjHandle, const uint64_t OffsetIn, const sec_cred_t ∗Ucred, const uint32_t BufferSize, const uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

This function will be removed in a future version of HPSS. Use hpss_ReadAttrsPlus().

Member hpss_ReaddirHandle (const ns_ObjHandle_t ∗ObjHandle, const uint64_t OffsetIn, const sec_cred_t ∗Ucred, const uint32_t BufferSize, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

This function will be removed in a future version of HPSS. Use hpss_ReaddirPlus().

Member hpss_ReadRawAttrsHandle (const ns_ObjHandle_t ∗ObjHandle, const uint64_t OffsetIn, const sec_cred_t ∗Ucred, const uint32_t BufferSize, const uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

This function will be removed in a future version of HPSS. Use hpss_ReadAttrsPlus().

Member hpss_SetLoginCred (const char ∗PrincipalName, hpss_authn_mech_t Mechanism, hpss_rpc_cred-_type_t CredType, hpss_rpc_auth_type_t AuthType, const void ∗Authenticator)

This function is deprecated in 10.1. Use hpss_SetAppLoginCred instead.

Member hpss_Statfs (uint32_t CosId, hpss_statfs_t ∗StatfsBuffer)

This function will be removed in a future version of HPSS. Use hpss_Statvfs64().

Member hpss_Statvfs (uint32_t CosId, hpss_statvfs_t ∗StatvfsBuffer)

This function will be removed in a future version of HPSS. Use hpss_Statvfs64.

Group math

Chapter 9

Module Documentation

9.1 File hash flag values

Variables

- const int BFS_FILE_HASH_DIGEST_FROM_USER = 0x0001
- const int BFS_FILE_HASH_DIGEST_GENERATED = 0x0008
- const int BFS_FILE_HASH_DIGEST_INVALID = 0x0010
- const int BFS_FILE_HASH_DIGEST_SKIP_VERF = 0x0004
- const int BFS_FILE_HASH_DIGEST_VERIFIED = 0x0002

9.1.1 Detailed Description

9.1.2 Variable Documentation

9.1.2.1 const int BFS_FILE_HASH_DIGEST_FROM_USER = 0x0001

This flag indicates that the digest was supplied by the end user

9.1.2.2 const int BFS_FILE_HASH_DIGEST_GENERATED = 0x0008

This flag provides a mechanism for the client application setting the hash digest to indicate that it actually generated the digest, as apposed to the case where it is handed the digest from the end user. This bit is only stored, and is not used internally by the hash digest verification logic.

9.1.2.3 const int BFS_FILE_HASH_DIGEST_INVALID = 0x0010

This flag indicates that an ∗ongoing∗ file hash has failed after closing the file. The client writing the data will have received an error on close, however in the case where they ignored that error and took no remedial action, we also mark that the digest was invalid at the time the data writes completed.

9.1.2.4 const int BFS_FILE_HASH_DIGEST_SKIP_VERF = 0x0004

This flag indicates that the end user requested that this hash not be verified on migration.

9.1.2.5 const int BFS_FILE_HASH_DIGEST_VERIFIED = 0x0002

This flag indicates that the digest was verified when the file was migrated

9.2 Client API

Definitions for the HPSS Client API.

Modules

- File Open, Create, and Close

  Functions which can be used to open and close HPSS files.
- File Data

  Functions which can be used to access and update HPSS file data.
- Fileset and Junction

  Functions which operate upon filesets and junctions.
- File Attribute

  Functions which operate upon HPSS file attributes.
- File Name

  Functions used to rename or remove names associated with files.
- ACLs

  Functions used to create, remove, and manipulate ACLs on files or directories.
- Directory Creation and Deletion

  Functions used to create, remove, and manipulate directories.
- Directory Data

  Functions used to access directory listings.
- Working Directory

  Functions used to determine or modify the working directory.
- Client API Control

  Functions used to control Client API state.
- Authentication

  Functions used to authenticate the Client API with HPSS.
- HPSS Statistics

  Functions used to retrieve or reset HPSS statistics.
- HPSS Object

  Functions used to manipulate or construct HPSS objects.
- Buffered Data

  Functions used to read and write data using a stream buffer.
- User-defined Attributes

  Functions used to update and access user-defined metadata.
- Data Structure Conversion

  Functions used to convert HPSS constructs to POSIX equivalents.
- Parallel I/O

  Functions which implement HPSS parallel I/O capabilities.
- Trashcan

  Functions which use the HPSS trashcan feature.
- POSIX APIs

  Posix Compliant HPSS APIs (Unsupported)
- Data Containers

  Functions which implement data containers.

Classes

- struct api_config

  Controls optional features of the Client API configuration. More...
- struct api_dist_file_info_t

  Distributed file information structure. More...
- struct api_namespec

  Name specifier information. More...
- struct hpss_errno_state

  HPSS Error Code State. More...
- struct hpss_global_fsent_t

  Global fileset entry. More...
- struct hpss_junction_ent_t

  HPSS junction entry Structure to hold junction information. More...
- struct hpss_pio_gapinfo_s

  PIO sparse file gap information. More...
- struct hpss_pio_params_s

  PIO I/O parameters. More...
- struct hpss_stat

  Stat structure for HPSS. More...
- struct hpss_statfs

  File system stats. More...
- struct hpss_statfs64

  File system stats (64bit) More...
- struct hpss_statvfs

  Virtual File System stat structure. More...
- struct hpss_statvfs64

  Virtual File System stat structure (64bit) More...
- struct hpss_userattr

  User-defined Attribute pair. More...
- struct hpss_userattr_list

  List of User-defined attribute pairs. More...
- struct subsys_stats

  HPSS Subystem statistics. More...

Macros

- #define API_CONFIG_ALL_FLAGS (0x0000001F)
- #define API_CONFIG_DEFAULT (0x00000000)
- #define API_DEBUG_ERROR (1)
- #define API_DEBUG_NONE (0)
- #define API_DEBUG_REQUEST (2)
- #define API_DEBUG_TRACE (4)
- #define API_DISABLE_CROSS_REALM (0x00000002)
- #define API_DISABLE_JUNCTIONS (0x00000004)
- #define API_ENABLE_LOGGING (0x00000001)
- #define API_MSGTYPE_MVRPROT MVRPROTOCOL_MSG_TYPE
- #define API_MSGTYPE_PDATA PDATA_MSG_TYPE
- #define API_TRANSFER_MVRSELECT 1
- #define API_TRANSFER_TCP 0
- #define API_USE_CONFIG (0x00000008)
- #define API_USE_SAN3P (0x00000010)

- #define HAVE_FSETFILEDIGEST 1

  HPSS file hash.
- #define HPSS_MIGRATE_FORCE (BFS_MIGRATE_FORCE)
- #define HPSS_MIGRATE_HPSS_ONLY (0x1)
- #define HPSS_MIGRATE_NO_COPY (BFS_MIGRATE_NO_COPY)
- #define HPSS_MIGRATE_PURGE_DATA (BFS_MIGRATE_PURGE_DATA)
- #define HPSS_O_EXCL (0x20000000)

  Open for exclusive access.
- #define HPSS_O_NO_TAPE (0x10000000)
- #define HPSS_O_STAGE_ASYNC (0x80000000)
- #define HPSS_O_STAGE_BKGRD (0x40000000)
- #define HPSS_O_STAGE_NONE (O_NONBLOCK)
- #define HPSS_PIO_HANDLE_GAP (1<<1)
- #define HPSS_PIO_PORT_RANGE (1<<2)
- #define HPSS_PIO_PUSH (1<<0)
- #define HPSS_PIO_USE_API_HOSTNAME (1<<3)
- #define HPSS_STAGE_STATUS_ACTIVE (2)
- #define HPSS_STAGE_STATUS_QUEUED (1)
- #define HPSS_STAGE_STATUS_UNKNOWN (0)
- #define MAX_XPATH_ELEM_SIZE (1001)
- #define MAX_XPATH_SIZE (MAX_XPATH_ELEM_SIZE∗125)
- #define UDA_API_VALUE 0
- #define UDA_API_XML 1
- #define XML_ATTR 1
- #define XML_NO_ATTR 2

Typedefs

- typedef struct api_config api_config_t

  Controls optional features of the Client API configuration.
- typedef struct api_namespec api_namespec_t

  Name specifier information.
- typedef struct hpss_errno_state hpss_errno_state_t

  HPSS Error Code State.
- typedef struct __HPSS_FILE HPSS_FILE
- typedef int(∗ hpss_pio_cb_t)(void ∗, uint64_t, unsigned int ∗, void ∗∗)

  Parallel IO callback.
- typedef struct hpss_pio_gapinfo_s hpss_pio_gapinfo_t

  PIO sparse file gap information.
- typedef void ∗ hpss_pio_grp_t

  Parallel IO group handle.
- typedef uint32_t hpss_pio_options_t
- typedef struct hpss_pio_params_s hpss_pio_params_t

  PIO I/O parameters.
- typedef struct hpss_stat hpss_stat_t

  Stat structure for HPSS.
- typedef struct hpss_statfs64 hpss_statfs64_t

  File system stats (64bit)
- typedef struct hpss_statfs hpss_statfs_t

  File system stats.
- typedef struct hpss_statvfs64 hpss_statvfs64_t

  Virtual File System stat structure (64bit)

- typedef struct hpss_statvfs hpss_statvfs_t

  Virtual File System stat structure.
- typedef struct hpss_userattr_list hpss_userattr_list_t

  List of User-defined attribute pairs.
- typedef struct hpss_userattr hpss_userattr_t

  User-defined Attribute pair.
- typedef struct subsys_stats subsys_stats_t

  HPSS Subystem statistics.

Enumerations

- enum hpss_pio_operation_t { HPSS_PIO_READ, HPSS_PIO_WRITE }
- enum hpss_pio_transport_t { HPSS_PIO_TCPIP, HPSS_PIO_MVR_SELECT }

  Parallel IO transport types.
- enum hpss_undelete_flags_t { HPSS_UNDELETE_NONE = 0, HPSS_UNDELETE_RESTORE_TIME, HPS-S_UNDELETE_OVERWRITE, HPSS_UNDELETE_OVERWRITE_AND_RESTORE }
- enum namespec_type_t { NAMESPEC_SKIP, NAMESPEC_REALM, NAMESPEC_USER, NAMESPEC_G-ROUP }

  Name specifier translation types.
- enum notrunc_flag_t { NOTRUNC_CLEAR = 0, NOTRUNC_SET }

  Truncation flags.
- enum purgelock_flag_t { PURGE_UNLOCK = 0, PURGE_LOCK, SUPER_PURGE_UNLOCK, SUPER_PU-RGE_LOCK }

  Purge lock flags Enumerate the different types of hpss_PurgeLock flag settings.
- enum purgeonmigrate_flag_t { PURGE_ON_MIGRATE_CLEAR = 0, PURGE_ON_MIGRATE_SET }

  Purge on Migrate flags.

Functions

- uint64_t API_AddAllRegisterValues (int LastPosition)

  Set all register bit positions up to the position provided.
- uint64_t API_AddRegisterValues (uint64_t InitialValue,...)

  Add values to a bit register.
- int API_GetServerIDForSubsystem (uint32_t SubsystemID, hpss_reqid_t RequestID, hpss_srvr_id_t ∗RetID)

  Retrieves the server id of the core server of a given subsystem.
- hpss_reqid_t API_GetUniqueRequestID ()

  Retrieves a unique request id.
- uint64_t API_RemoveRegisterValues (uint64_t InitialValue,...)

  Remove values from a bit register.
- int API_StageBatchGetEntry (hpss_stage_batch_t ∗Batch, int Idx, hpss_stage_t ∗Entry)

  Retrieve Stage Batch Entry.
- int API_StageBatchInsertBFHandle (hpss_stage_batch_t ∗Batch, int Idx, hpss_object_handle_t ∗ObjHandle, int FromStorageLevel, int ToStorageLevel, u_signed64 Offset, u_signed64 Length, uint32_t Flags)

  Set a Stage Batch Entry using a Bitfile Handle.
- int API_StageBatchInsertBFObj (hpss_stage_batch_t ∗Batch, int Idx, const bfs_bitfile_obj_handle_t ∗BFObj, int FromStorageLevel, int ToStorageLevel, u_signed64 Offset, u_signed64 Length, uint32_t Flags)

  Set a Stage Batch Entry using a Bitfile ID.
- int API_StageBatchInsertFd (hpss_stage_batch_t ∗Batch, int Idx, int Fd, int FromStorageLevel, int ToStorage-Level, u_signed64 Offset, u_signed64 Length, uint32_t Flags)

  Set a Stage Batch Entry using a File Descriptor.
- int hpss_AbortIOByMoverId (int32_t SubsystemId, hpss_srvr_id_t MoverId)

Abort all I/O requests for the specified mover ID.
- int hpss_AbortIOByRequestId (int32_t SubsystemId, hpss_reqid_t RequestIdToAbort, sec_cred_t ∗Ucred)

    Abort a specific I/O request.
- int hpss_AbortIOByRequestMatch (int32_t SubsystemId, char ∗PartialReqString, sec_cred_t ∗Ucred)

    Abort an I/O request that matches the request string.
- int hpss_AbortIOByUserId (int32_t SubsystemId, sec_cred_t ∗Ucred, uint32_t ∗RequestsAborted, uint32_t ∗RequestsAlreadyAborted, uint32_t ∗RequestsFound)

    Abort all I/O requests for the specified user.
- int hpss_AcctCodeToName (acct_rec_t AcctCode, hpss_id_t ∗Site, char ∗AcctName)

    Maps an account code to its corresponding account name.
- int hpss_AcctNameToCode (char ∗AcctName, hpss_id_t ∗Site, acct_rec_t ∗AcctCode)

    Maps an account name to its corresponding account code.
- char ∗ hpss_BuildLevelString (void)

    Construct a string containing HPSS version information.
- void hpss_ClearLastHPSSErrno ()

    Clears the current HPSS errno state.
- int hpss_ConvertIdsToNames (int32_t NumEntries, api_namespec_t ∗Specs)

    Converts user or group ids to names.
- int hpss_ConvertNamesToIds (int32_t NumEntries, api_namespec_t ∗Specs)

    Converts user or group names to ids.
- char ∗∗ hpss_CopyrightStrings (void)

    Read the copyright file and return the strings.
- int hpss_FdelFileDigestList (int Fildes, hpss_file_hash_stripe_flags_t Flags)

    Delete file hash digest information.
- int hpss_FgetFileDigest (int Fildes, hpss_file_hash_digest_t ∗Digest)

    Retrieves the file's hash digest, if any, for an open file.
- int hpss_FgetFileDigestList (int Fildes, hpss_file_hash_stripe_flags_t Flags, uint64_t ∗StripeLength, hpss_-file_hash_digest_list_t ∗DigestList)

    Retrieves a file's hash digest, if any, for an open file.
- int hpss_FileGetDigestHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, sec_cred_t ∗Ucred, hpss_file_hash_digest_t ∗Digest)

    Retrieves the file's hash digest, if any.
- int hpss_FileGetDigestListHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, sec_cred_t ∗Ucred, hpss_file_hash_stripe_flags_t Flags, uint64_t ∗StripeLength, hpss_file_hash_digest_list_t ∗DigestList)

    Retrieves a file's hash digest(s) by preference, if any.
- int hpss_FsetFileDigest (int Fildes, const hpss_file_hash_digest_t ∗Digest)

    Sets single stripe file's hash digest information.
- int hpss_FsetFileDigestList (int Fildes, uint64_t StripeLength, const hpss_file_hash_digest_list_t ∗DigestList)

    Sets file hash digest information.
- int hpss_GetAcct (acct_rec_t ∗RetDefAcct, acct_rec_t ∗RetCurAcct)

    Retrieves current and default account codes.
- int hpss_GetAcctName (char ∗AcctName)

    Retrieves account name.
- int hpss_GetAllSubsystems (ls_map_array_t ∗∗ls_map_array)

    Retrieve a map of all subsystems.
- int hpss_GetAsyncStatus (hpss_reqid_t CallBackId, bfs_bitfile_obj_handle_t ∗BitfileObj, int32_t ∗Status)

    Get the status of a background stage.
- int hpss_GetBatchAsynchStatus (hpss_reqid_t CallBackId, hpss_stage_bitfile_list_t ∗BFObjs, hpss_stage_-status_type_t Type, hpss_stage_batch_status_t ∗Status)

    Check the status of files in a batch request.
- int hpss_GetDistFile (int Fildes, api_dist_file_info_t ∗FileInfo)

Extracts a file table entry for an open file descriptor.

- int hpss_GetFullPath (const ns_ObjHandle_t ∗ObjHandle, char ∗∗FullPath)

  Provides the full path back to an object from the root of roots.
- int hpss_GetFullPathBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, char ∗∗FullPath)

  Provides a full path back to a bitfile from the root of roots.
- int hpss_GetFullPathBitfileLen (const bfs_bitfile_obj_handle_t ∗BitfileObj, char ∗Path, size_t BufLen)

  Provides a full path back to a bitfile from the root of roots.
- int hpss_GetFullPathHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, char ∗∗FullPath)

  Provides the full path back to an object from the root of roots.
- int hpss_GetFullPathHandleLen (const ns_ObjHandle_t ∗ObjHandle, const char ∗ObjPath, char ∗Path, size_t BufLen)

  Provides the full path back to an object from the root of roots.
- int hpss_GetFullPathLen (const ns_ObjHandle_t ∗ObjHandle, char ∗Path, size_t BufLen)

  Provides the full path back to an object from the root of roots.
- int32_t hpss_GetGID (gid_t ∗GID)

  Get the group id for the calling thread.
- hpss_errno_state_t hpss_GetLastHPSSErrno ()

  Retrieves the error state of the last HPSS error.
- hpss_reqid_t hpss_GetNextIORequestID ()

  Retrieves the next request id that will be used for I/O.
- void hpss_HashFlagsToString (unsigned16 flags, char ∗buf, int len)

  Returns a string representation of the provided hash flags.
- int hpss_InsertDistFile (const api_dist_file_info_t ∗FileInfo)

  Inserts a file table entry into the current file table.
- int hpss_LoadThreadState (uid_t UserID, mode_t Umask, const char ∗ClientFullName)

  Allows some clients to manipulate the local state of a thread.
- int hpss_LookupRootCS (const char ∗SiteName, hpss_srvr_id_t ∗ServerId)

  Retrieves the ID for a site's root Core Server.
- int hpss_Migrate (int Fildes, uint32_t SrcLevel, uint32_t Flags, uint64_t ∗RetBytesMigrated)

  Migrates data in an open file from a specified hierarchy level.
- int hpss_PingCore (const hpss_srvr_id_t ∗CoreServerID, uint32_t ∗RecvSecs, uint32_t ∗RecvUSecs)

  Pings the Core Server.
- int hpss_Purge (int Fildes, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, uint32_t Flags, uint64_t ∗RetBytesPurged)

  Purges data in an open file from a specified hierarchy level.
- int hpss_SetAcct (acct_rec_t NewCurAcct)

  Sets the current account code.
- int hpss_SetAcctByName (const char ∗NewAcctName)

  Sets the current account name.
- int hpss_SetGID (uint32_t NewCurGid)

  Set the group id for the calling thread.
- int hpss_SiteIdToName (const hpss_id_t ∗SiteId, char ∗SiteName)

  Convert Site ID to Site Name.
- int hpss_SiteNameToId (const char ∗SiteName, hpss_id_t ∗SiteId)

  Convert Site Name to Site ID.
- int hpss_Stage (int Fildes, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, uint32_t Flags)

  Stage an open HPSS file.
- int hpss_StageBatch (hpss_stage_batch_t ∗Batch, hpss_stage_batch_status_t ∗Status)

  Stage a batch of bitfiles.
- int hpss_StageBatchCallBack (hpss_stage_batch_t ∗Batch, bfs_callback_addr_t ∗CallBackPtr, hpss_reqid_t ∗ReqID, hpss_stage_bitfile_list_t ∗BFObjsList, hpss_stage_batch_status_t ∗Status)

Stage a batch of bitfiles asynchronously.

- int hpss_StageCallBack (const char ∗Path, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, bfs_-callback_addr_t ∗CallBackPtr, uint32_t Flags, hpss_reqid_t ∗ReqID, bfs_bitfile_obj_handle_t ∗BitfileObj)

  Stage an HPSS file in the background.

- int hpss_StageCallBackBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, bfs_callback_addr_t ∗CallBackPtr, uint32_t Flags, hpss_reqid_t ∗ReqID)

  Stage a file in the background using its bitfile id.

- mode_t hpss_Umask (mode_t CMask)

  Set the file mode creation mask, and returns the previous mask value.

- int hpss_UserAttrListAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, hpss_userattr_list_t ∗Attrs, int Flags, int XMLSize)

  List User-defined Attributes associated with a namespace path using a handle.

9.2.1 Detailed Description

Definitions for the HPSS Client API.

9.2.2 Class Documentation

9.2.2.1 struct api_config

Controls optional features of the Client API configuration.

Class Members

char	Application[HPSS_MAX_APP_NAME]	Application name (for log messages)
hpss_authn_mech_t	AuthnMech	Authentication mechanism to use (Unix or Kerberos)
int	BusyDelay	Number of seconds to delay between retry attempts
int	BusyRetries	Number of retries to perform due to server busy Used to control the number of retries to be performed when a request fails because the Core Server does not currently have an available thread to handle that request. A value of zero indicates that no retries are to be performed. A value of negative one indicates that retries should be performed until either the request succeeds or fails for another reason.
char	DebugPath[HPSS_MAX_PATH_NAME]	Debug log path If generation of debug messages is enabled then this is the path name of the file to which log messages will be directed. Special cases are "stdout" and "stderr".
int	DebugValue	Level of debug output. Specifies the level of debug output; if zero, indicates that the Client API will not send debug messages. The following levels of debug may be set and bitwise combined: • API_DEBUG_ERROR - Error logging • API_DEBUG_REQUEST - Request logging • API_DEBUG_TRACE - Trace logging Note that some errors that are logged are informational and do not indicate a problem with the system.
char	DescName[HPSS_MAX_DESC_NAME]	Name to use when generating HPSS log messages **Deprecated** Remove in next major release.
int	DMAPWriteUpdates	Defunct field **Deprecated** This field has been unused since HPSS 6.
unsigned int	Flags	Bitmap of configuration flags Bitmap of configuration flags, including: • API_ENABLE_LOGGING - If logging compiled into Client API library, perform logging on errors • API_USE_CONFIG - Use configuration from API • API_DISABLE_CROSS_REALM - If set, this flag prevents the Client API from contacting any servers outside of the local relam. Once set, this flag cannot be unset. • API_DISABLE_JUNCTIONS - If set, prevents the Client API from processing any requests which require it to traverse a junction. Once set this flag cannot be unset.

char	HostName[HPSS_MAX_HOST_NAME]	Control the host interface Specifies interface name to use for TCP/IP communication
int	LimitedRetries	Control limited retry count Used to control the number of retry attempts for limited retry type errors
int	MaxConnections	Maximum server connections allowed Maximum number of connections for use by the connection management module
int	NumRetries	Number of retries for failed operations Control the number of retries when an operation fails. Zero means that no retries are done. -1 means that retries are done until the operation is successful.
int	RetryStageInp	Controls background stage on open behavior Control whether retries are attempted on file opens in a COS that is configured for background staging on open. A non-zero value indicates that an open should return -EINPROGRESS to indicate that the file being staged will be retried. A value of zero indicates that the -EINPROGRESS should be returned to the client.
int	ReuseDataConnections	Reuse I/O connections for each file Used to control whether TCP/IP connections should be left open as long as a file is opened or closed after each I/O request. A non-zero value will cause connections to remain open while a zero will cause connections to be closed.
hpss_rpc_prot_level_t	RPCProtLevel	Level of RPC protection to use
int	TotalDelay	Number of seconds to retry Control the number of total seconds to continue retrying a request.
int	TransferType	Transfer protocol mechanism Indicates the type of transfer mechanism to be used: • API_TRANSFER_TCP - Use TCP/IP • API_TRANSFER_MVRSELECT - Let the Mover select the transfer protocol
int	UsePortRange	Use the configured port range for I/O Control whether the HPSS mover(s) should use the configured port range when making TCP/IP connections for read and write requests. A non-zero value will cause the mover(s) to use the port range. A zero will cause the mover(s) to allow the operating system to select the port number.
uint32_t	XMLSize	Default XML size to use for UDA operations

9.2.2.2 struct api_dist_file_info_t

Distributed file information structure.

Class Members

uchar	CkSum	checksum digest
struct file_info	Info	Distributed file information

9.2.2.3 struct api_namespec

Name specifier information.

Structures that contain information to translate between user names and UIDs or group names and GIDs.

Class Members

uint32_t	Id	Principal's uid or gid
char	Name[HPSS_M-AX_PRINCIPAL-_NAME]	Name of principal
uint32_t	RealmId	HPSS Realm Id where principal resides
char	RealmName[HP-SS_MAX_REAL-M_NAME]	Name of realm where principal resides
namespec_type-_t	Type	Type of name represented

9.2.2.4 struct hpss_errno_state

HPSS Error Code State.

The hpss_errno_state_t structure is used to maintain the state of a previous error reported by HPSS. Error code meanings can be determined by consulting the hpss_errno.h file.

Class Members

char	func[HPSS_MA-X_PATH_NAM-E]	HPSS function which returned the error
int32_t	hpss_errno	Error code reported by HPSS
hpss_reqid_t	requestId	Request id which triggered the error

9.2.2.5 struct hpss_global_fsent_t

Global fileset entry.

Class Members

hpss_srvr_id_t	CoreServerId	Default Core Server
uint64_t	FilesetId	Fileset identifier Fileset identifier
unsigned char	FilesetName[H-PSS_MAX_FS_-NAME_LENGT-H]	Fileset name Fileset name

9.2.2.6 struct hpss_junction_ent_t

HPSS junction entry Structure to hold junction information.

Class Members

ns_ObjHandle_t	FilesetHandle	Junction fileset handle
ns_ObjHandle_t	JunctionHandle	Junction object handle
unsigned char	JunctionPath-Name[HPSS_M-AX_PATH_NA-ME]	Junction path

9.2.2.7 struct hpss_pio_gapinfo_s

PIO sparse file gap information.

Class Members

uint64_t	Length	Length of gap
uint64_t	Offset	Starting offset of gap

9.2.2.8 struct hpss_pio_params_s

PIO I/O parameters.

Class Members

uint32_t	BlockSize	Bytes for each element
uint32_t	ClntStripeWidth	Number of elements in client stripe
uint32_t	FileStripeWidth	Number of elements in file stripe
uint32_t	IOTimeOutSecs	Seconds to wait for I/O. Setting this to zero gets the default time (30 minutes)
hpss_pio_-operation_t	Operation	Operation type for PIO group
hpss_pio_-options_t	Options	PIO data transport options
hpss_pio_-transport_t	Transport	Type of data transport to use

9.2.2.9 struct hpss_stat

Stat structure for HPSS.

Class Members

timestamp_sec-_t	hpss_st_atime	Time of last access
timestamp_sec-_t	hpss_st_-birthtime	Time of file create
timestamp_sec-_t	hpss_st_ctime	Time of last file status change
timestamp_sec-_t	hpss_st_mtime	Time of last data modification
uint32_t	st_blksize	Optimal blocksize for file system I/O ops
uint32_t	st_blocks	Actual number of blocks allocated in DEV_BSIZE blocks
uint32_t	st_dev	ID of device containing a directory entry for this file. File serial no + device ∗ ID uniquely identify the file within the system
uint16_t	st_flag	Flag
uint32_t	st_gen	Inode generation number
uint32_t	st_gid	Group ID of the file's group
uint64_t	st_ino	File serial number
uint32_t	st_mode	File mode
uint16_t	st_nlink	Number of links
uint32_t	st_rdev	ID of the device. This entry is defined only for character or block special files.

uint64_t	st_size	File size
uint64_t	st_ssize	64-bit file size
uint32_t	st_type	Vnode type
uint32_t	st_uid	User ID of the file's owner
uint32_t	st_vfs	Vfs number
int32_t	st_vfstype	Type of fs (see vnode.h)

9.2.2.10 struct hpss_statfs

File system stats.

Class Members

uint32_t	f_bavail	Free blocks available
uint32_t	f_bfree	Free blocks in file system
uint32_t	f_blocks	Total data blocks in the file system
uint32_t	f_bsize	Preferred file system block size
uint32_t	f_ffree	Total number of free file nodes
uint32_t	f_files	Total number of file nodes
char	f_fname[32]	File system name (usually a mount point)
char	f_fpack[32]	File system pack name
uint32_t	f_fsid	File system id
uint32_t	f_namemax	Maximum filename length

9.2.2.11 struct hpss_statfs64

File system stats (64bit)

Class Members

uint64_t	f_bavail	Free blocks available
uint64_t	f_bfree	Free blocks in file system
uint64_t	f_blocks	Total data blocks in the file system
uint64_t	f_bsize	Preferred file system block size
uint64_t	f_ffree	Total number of free file nodes
uint64_t	f_files	Total number of file nodes
char	f_fname[32]	File system name (usually a mount point)
char	f_fpack[32]	File system pack name
uint32_t	f_fsid	File system id
uint32_t	f_namemax	Maximum filename length

9.2.2.12 struct hpss_statvfs

Virtual File System stat structure.

Class Members

uint32_t	f_bavail	Free blocks available
uint32_t	f_bfree	Free blocks in file system

uint32_t	f_blocks	Total data blocks in file system
uint32_t	f_bsize	Preferred file system block size
uint32_t	f_ffree	Free file nodes in file system
uint32_t	f_files	Total file nodes in file system
char	f_fpack[32]	File system pack name
uint32_t	f_fsid	File system id
char	f_fstr[32]	File system specific string
uint32_t	f_namemax	Maximum filename length

9.2.2.13 struct hpss_statvfs64

Virtual File System stat structure (64bit)

Class Members

uint64_t	f_bavail	Free blocks available
uint64_t	f_bfree	Free blocks in file system
uint64_t	f_blocks	Total data blocks in file system
uint64_t	f_bsize	Preferred file system block size
uint64_t	f_ffree	Free file nodes in file system
uint64_t	f_files	Total file nodes in file system
char	f_fpack[32]	File system pack name
uint32_t	f_fsid	File system id
char	f_fstr[32]	File system specific string
uint32_t	f_namemax	Maximum filename length

9.2.2.14 struct hpss_userattr

User-defined Attribute pair.

Prototypes and Data Structures for User-defined Attributes

Class Members

char ∗	Key	Attribute name
char ∗	Value	Attribute value

9.2.2.15 struct hpss_userattr_list

List of User-defined attribute pairs.

Class Members

int	len	Length of Pair
hpss_userattr_t ∗	Pair	Array of userattr pairs

9.2.2.16 struct subsys_stats

HPSS Subystem statistics.

Class Members

uint32_t	COSChange-ThreadCount	Number of COS change threads in the Core Server
uint32_t	DeleteCount	Files deleted since last reset
uint32_t	MigrationCount	Files migrated since last reset
uint32_t	PurgeCount	Files purged since last reset
uint32_t	StageCount	Files staged since last reset
timestamp_sec-_t	TimeLastReset	Time of last statistics reset

9.2.3 Macro Definition Documentation

9.2.3.1 #define API_CONFIG_ALL_FLAGS (0x0000001F)

All valid flags

9.2.3.2 #define API_CONFIG_DEFAULT (0x00000000)

Default config flags

9.2.3.3 #define API_DEBUG_ERROR (1)

Error code level debugging

9.2.3.4 #define API_DEBUG_NONE (0)

No debugging

9.2.3.5 #define API_DEBUG_REQUEST (2)

Request level debugging

9.2.3.6 #define API_DEBUG_TRACE (4)

Trace level debugging

9.2.3.7 #define API_DISABLE_CROSS_REALM (0x00000002)

Disable cross-realm (defunct)

9.2.3.8 #define API_DISABLE_JUNCTIONS (0x00000004)

Disable junction traversal

9.2.3.9 #define API_ENABLE_LOGGING (0x00000001)

Enable HPSS logging

9.2.3.10 #define API_MSGTYPE_MVRPROT MVRPROTOCOL_MSG_TYPE

Mover protocol type

9.2.3.11 #define API_MSGTYPE_PDATA PDATA_MSG_TYPE

PDATA protocol type

9.2.3.12 #define API_TRANSFER_MVRSELECT 1

Mover selects the protocol

9.2.3.13 #define API_TRANSFER_TCP 0

Use the TCP PDATA protocol

9.2.3.14 #define API_USE_CONFIG (0x00000008)

Use the provided API configuration

9.2.3.15 #define API_USE_SAN3P (0x00000010)

Enable SAN3P transfers

9.2.3.16 #define HPSS_MIGRATE_FORCE (BFS_MIGRATE_FORCE)

If on, migrate data even when there is already a copy at a lower level

9.2.3.17 #define HPSS_MIGRATE_HPSS_ONLY (0x1)

If on, don’t migrate dmap

9.2.3.18 #define HPSS_MIGRATE_NO_COPY (BFS_MIGRATE_NO_COPY)

If on, do not migrate multiple copies

9.2.3.19 #define HPSS_MIGRATE_PURGE_DATA (BFS_MIGRATE_PURGE_DATA)

If on, purge data after migration is complete

9.2.3.20 #define HPSS_O_EXCL (0x20000000)

Open for exclusive access.

Open a file for exclusive access. This is only valid for bitfiles. An open for exclusive access will fail if any other activity is already occurring on the bitfile. While opened for exclusive access, no other thread or application may open the bitfile.

9.2.3.21 #define HPSS_O_NO_TAPE (0x10000000)

Do not use tape

9.2.3.22 #define HPSS_O_STAGE_ASYNC (0x80000000)

Stage asynchrously on open

9.2.3.23 #define HPSS_O_STAGE_BKGRD (0x40000000)

Stage in the background

9.2.3.24 #define HPSS_O_STAGE_NONE (O_NONBLOCK)

Do not stage on open

9.2.3.25 #define HPSS_PIO_HANDLE_GAP (1<<1)

PIO will handle file gaps

9.2.3.26 #define HPSS_PIO_PORT_RANGE (1<<2)

Use specified port range

9.2.3.27 #define HPSS_PIO_PUSH (1<<0)

Use the PDATA PUSH protocol

9.2.3.28 #define HPSS_PIO_USE_API_HOSTNAME (1<<3)

Listen on the API_HOSTNAME interface by default

9.2.3.29 #define HPSS_STAGE_STATUS_ACTIVE (2)

Stage request is being processed

9.2.3.30 #define HPSS_STAGE_STATUS_QUEUED (1)

Stage request is queued in the bitfile layer

9.2.3.31 #define HPSS_STAGE_STATUS_UNKNOWN (0)

Stage state is unknown

9.2.3.32 #define MAX_XPATH_ELEM_SIZE (1001)

Maximum size of an XML element

9.2.3.33 #define MAX_XPATH_SIZE (MAX_XPATH_ELEM_SIZE*125)

Maximum XPath size

9.2.3.34 #define UDA_API_VALUE 0

Provide the text value only

9.2.3.35 #define UDA_API_XML 1

Provide the ML

9.2.3.36 #define XML_ATTR 1

Include XML attribute fields

9.2.3.37 #define XML_NO_ATTR 2

Do not include XML attribute fields

9.2.4 Typedef Documentation

9.2.4.1 typedef struct api_namespec api_namespec_t

Name specifier information.

Structures that contain information to translate between user names and UIDs or group names and GIDs.

9.2.4.2 typedef struct hpss_errno_state hpss_errno_state_t

HPSS Error Code State.

The hpss_errno_state_t structure is used to maintain the state of a previous error reported by HPSS. Error code meanings can be determined by consulting the hpss_errno.h file.

9.2.4.3 typedef struct __HPSS_FILE HPSS_FILE

HPSS file stream

9.2.4.4 typedef uint32_t hpss_pio_options_t

Parallel IO transport options

9.2.4.5 typedef struct hpss_userattr hpss_userattr_t

User-defined Attribute pair.

Prototypes and Data Structures for User-defined Attributes

9.2.5 Enumeration Type Documentation

9.2.5.1 enum hpss_pio_operation_t

Parallel IO operation types

Enumerator

HPSS_PIO_READ	Read Operation
HPSS_PIO_WRITE	Write Operation

9.2.5.2 enum hpss_pio_transport_t

Parallel IO transport types.

Enumerator

HPSS_PIO_TCPIP	Use TCP/IP
HPSS_PIO_MVR_SELECT	Allow the Mover to select the protocol

9.2.5.3 enum hpss_undelete_flags_t

Undelete control flags

Enumerator

HPSS_UNDELETE_NONE	no options
HPSS_UNDELETE_RESTORE_TIME	restore timestamps
HPSS_UNDELETE_OVERWRITE	overwrite an existing file
HPSS_UNDELETE_OVERWRITE_AND_RESTORE	overwrite + restore

9.2.5.4 enum namespec_type_t

Name specifier translation types.

Enumerator

NAMESPEC_SKIP	Place holder; no translation needed.
NAMESPEC_REALM	Translate realm name only
NAMESPEC_USER	Translate user name and realm name
NAMESPEC_GROUP	Translate group name and realm name

9.2.5.5 enum notrunc_flag_t

Truncation flags.

Enumerate the different values for the Flag for hpss_SetFileNotrunc.

Enumerator

NOTRUNC_CLEAR	Clear the don't truncate final segment flag
NOTRUNC_SET	Set the don't truncate final segment flag

9.2.5.6 enum purgelock_flag_t

Purge lock flags Enumerate the different types of hpss_PurgeLock flag settings.

The purpose of the BFS_NO_PURGE flag (purge lock) is to lock the file at level 0 disk so it cannot be purged prematurely, before the application which wrote or staged it is finished with it.

The purpose of the BFS_SUPER_NO_PURGE flag (super purge lock) is protect data integrity by locking every level of the file, disk or tape, in order to prevent the accidental deletion of the last copy of the file when we have violated our own rules by a force purge.

The super purge lock may be set or cleared only by an authorized user (one who has control permission on the core server's ACL).

Enumerator

PURGE_UNLOCK	Purge unlock the file
PURGE_LOCK	Purge lock the file
SUPER_PURGE_UNLOCK	Super purge unlock the file (must be an authorized user)
SUPER_PURGE_LOCK	Super purge lock the file (must be an authorized user)

9.2.5.7 enum purgeonmigrate_flag_t

Purge on Migrate flags.

Enumerate the different values for the Flag for hpss_PurgeOnMigrate.

Enumerator

PURGE_ON_MIGRATE_CLEAR	Clear the purge on migrate flag
PURGE_ON_MIGRATE_SET	Set the purge on migrate flag

9.2.6 Function Documentation

9.2.6.1 uint64_t API_AddAllRegisterValues (int *LastPosition*)

Set all register bit positions up to the position provided.

Parameters

`in`	*LastPosition*	Last position to turn on

This routine turns on all bit positions from zero to the value passed in *LastPosition*.

Returns

New 64-bit value will all bit positions up to *LastPosition* set

See Also

API_AddRegisterValues, API_RemoveRegisterValues

9.2.6.2 uint64_t API_AddRegisterValues (uint64_t *InitialValue*, ...)

Add values to a bit register.

Parameters

`in`	*InitialValue*	Initial value
`in`	...	A series of bit positions to turn on within the provided *InitialValue*. Terminate with -1.

This routine OR's a number of 64-bit values together, given the bit positions to turn on. The list of values is terminated by a value of -1.

Returns

New 64-bit value with all bit positions in the series turned on.

See Also

API_AddAllRegisterValues, API_RemoveRegisterValues

9.2.6.3 int API_GetServerIDForSubsystem (uint32_t *SubsystemID*, hpss_reqid_t *RequestID*, hpss_srvr_id_t * *RetID*)

Retrieves the server id of the core server of a given subsystem.

Get the server id of core server for the specified subsystem identifier.

Parameters

`in`	*SubsystemID*	subsystem id
`in`	*RequestID*	request id
`out`	*RetID*	returned ID

Return values

-EINVAL	More than one Core Server is configuredfor the subsystem.
-ENOENT	No Core Servers are configured for the subsystem.
0	No error
Other	Error returned by hpss_LocateServersByType()or hpss_LocateRootCS()

9.2.6.4 hpss_reqid_t API_GetUniqueRequestID ( )

Retrieves a unique request id.

The API_GetUniqueRequestID will get a request id that is unique to this machine.

Returns

The request id, a unique identifier.

9.2.6.5 uint64_t API_RemoveRegisterValues (uint64_t *InitialValue*, ...)

Remove values from a bit register.

Parameters

`in`	*InitialValue*	Initial Value
`in`	...	A series of bit positions to turn off within the provided *InitialValue*. Terminate with -1.

This routine AND's the complement of a multiple number of 64-bit values with an initial value, *InitialValue*, given the bit positions to turn off. The result of the operation is also returned. The list of values to turn off is terminated by a value of -1.

Returns

New 64-bit value with all bit positions in the provided series turned off.

See Also

API_AddAllRegisterValues, API_AddRegisterValues

9.2.6.6 int API_StageBatchGetEntry (hpss_stage_batch_t ∗ *Batch*, int *Idx*, hpss_stage_t ∗ *Entry*)

Retrieve Stage Batch Entry.

Parameters

`in`	*Batch*	Batch Request Structure
`in`	*Idx*	Batch List Index
`out`	*Entry*	Stage Batch Entry Structure

Return values

HPSS_E_NOERROR	Success
HPSS_EFAULT	NULL Batch Structure Provided
HPSS_EFAULT	NULL Entry Pointer Provided
HPSS_EINVAL	Invalid List Index Provided
HPSS_EFAULT	No Batch List allocated

9.2.6.7 int API_StageBatchInsertBFHandle (hpss_stage_batch_t ∗ *Batch,* int *Idx,* hpss_object_handle_t ∗ *ObjHandle,* int *FromStorageLevel,* int *ToStorageLevel,* u_signed64 *Offset,* u_signed64 *Length,* uint32_t *Flags*)

Set a Stage Batch Entry using a Bitfile Handle.

Parameters

`in`	*Batch*	Batch Request Structure
`in`	*Idx*	Batch List Index
`in`	*ObjHandle*	Bitfile Handle
`in`	*FromStorage-Level*	Level to Stage Data from
`in`	*ToStorageLevel*	Level to Stage Data to
`in`	*Offset*	Data Offset to Stage
`in`	*Length*	Length of Data to Stage
`in`	*Flags*	Stage Flags

Return values

HPSS_E_NOERROR	Success
HPSS_EFAULT	NULL Batch Structure Provided
HPSS_EFAULT	NULL Object Handle Provided
HPSS_EFAULT	No Batch List allocated
HPSS_EINVAL	Invalid List Index Provided

Note

Valid flags:

- BFS_STAGE_ALL (Stage All Data)
- BFS_NO_FAR (Disable Full Aggregate Recall)
- BFS_NO_TAPE (Do not stage from tape)

9.2.6.8 int API_StageBatchInsertBFObj (hpss_stage_batch_t ∗ *Batch,* int *Idx,* const bfs_bitfile_obj_handle_t ∗ *BFObj,* int *FromStorageLevel,* int *ToStorageLevel,* u_signed64 *Offset,* u_signed64 *Length,* uint32_t *Flags*)

Set a Stage Batch Entry using a Bitfile ID.

Parameters

`in,out`	*Batch*	Batch Request Structure
`in`	*Idx*	Batch List Index
`in`	*BFObj*	Bitfile Obj
`in`	*FromStorage-Level*	Level to Stage Data from
`in`	*ToStorageLevel*	Level to Stage Data to
`in`	*Offset*	Data Offset to Stage
`in`	*Length*	Length of Data to Stage
`in`	*Flags*	Stage Flags

Return values

HPSS_E_NOERROR	Success
HPSS_EFAULT	NULL Batch Structure Provided
HPSS_EFAULT	NULL Bitfile ID Provided
HPSS_EFAULT	No Batch List allocated
HPSS_EINVAL	Invalid List Index Provided

Note

Valid flags:

- BFS_STAGE_ALL (Stage All Data)
- BFS_NO_FAR (Disable Full Aggregate Recall)
- BFS_NO_TAPE (Do not stage from tape)

9.2.6.9 int API_StageBatchInsertFd (hpss_stage_batch_t ∗ *Batch*, int *Idx*, int *Fd*, int *FromStorageLevel*, int *ToStorageLevel*, u_signed64 *Offset*, u_signed64 *Length*, uint32_t *Flags*)

Set a Stage Batch Entry using a File Descriptor.

Parameters

`in`	*Batch*	Batch Request Structure
`in`	*Idx*	Batch List Index
`in`	*Fd*	Open File Descriptor
`in`	*FromStorage-Level*	Level to Stage Data from
`in`	*ToStorageLevel*	Level to Stage Data to
`in`	*Offset*	Data Offset to Stage
`in`	*Length*	Length of Data to Stage
`in`	*Flags*	Stage Flags

Return values

HPSS_E_NOERROR	Success
HPSS_EFAULT	NULL Batch Structure Provided
HPSS_EFAULT	No Batch List allocated
HPSS_EINVAL	Invalid List Index Provided

Note

Valid flags:

- BFS_STAGE_ALL (Stage All Data)
- BFS_NO_FAR (Disable Full Aggregate Recall)
- BFS_NO_TAPE (Do not stage from tape)

9.2.6.10 int hpss_AbortIOByMoverId (int32_t *SubsystemId*, hpss_srvr_id_t *MoverId*)

Abort all I/O requests for the specified mover ID.

This routine is called to request an abort for all I/O requests for the specified mover ID. That is, the server ID of the Mover.

Parameters

`in`	*SubsystemId*	Subsystem Id
`in`	*MoverId*	Mover Server Id

Return values

0	Success
Other	Error Code indicating the nature of the problem

Note

This API is only usable by empowered users. Non-empowered users will receive an error.

9.2.6.11 int hpss_AbortIOByRequestId (int32_t *SubsystemId*, hpss_reqid_t *RequestIdToAbort*, sec_cred_t ∗ *Ucred*)

Abort a specific I/O request.

This routine is called to request an abort of a specific I/O request.

Parameters

`in`	*SubsystemId*	Subsystem ID
`in`	*RequestIdTo-Abort*	Request ID
`in`	*Ucred*	User Credentials (optional)

Return values

0	Success
Other	Error Code indicating the nature of the problem

9.2.6.12 int hpss_AbortIOByRequestMatch (int32_t *SubsystemId*, char ∗ *PartialReqString*, sec_cred_t ∗ *Ucred*)

Abort an I/O request that matches the request string.

This routine is called to request an abort of a specific I/O request.

Parameters

`in`	*SubsystemId*	Subsystem ID
`in`	*PartialReqString*	Partial request ID to abort
`in`	*Ucred*	User Credentials (optional)

Return values

0	Success
HPSS_EFAULT	No request string was provided
HPSS_EINVAL	More than one match
HPSS_RANGE	Partial request string was too long
Other	Error Code indicating the nature of the problem

Note

> Dangers of this API. This API will abort based on a partial request ID string of length >= 1. It is up to the caller to enforce any additional limitations. This API will abort any request whose request id matches the partial request string uniquely (e.g. there is only one match). This means that it is possible that the use of this API could inadvertantly abort a request if two matches existed when the partial request string was found but only one (not the target) existed when the API is processed.

For safest use, use the partial request ID soon after obtaining it, and do not use with empowered users.

Note

> This API is also slower than the exact match API because it has to scan the entire job table rather than doing a hashtable lookup. It is inefficient for use in large batch processing.

9.2.6.13 int hpss_AbortIOByUserId (int32_t *SubsystemId,* sec_cred_t ∗ *Ucred,* uint32_t ∗ *RequestsAborted,* uint32_t ∗ *RequestsAlreadyAborted,* uint32_t ∗ *RequestsFound*)

Abort all I/O requests for the specified user.

This routine is called to request an abort for all I/O requests for the specified user.

Parameters

in	*SubsystemId*	Subsystem ID
in	*Ucred*	User Credentials
out	*Requests-Aborted*	Requests Aborted
out	*Requests-AlreadyAborted*	Requests Already Aborted
out	*RequestsFound*	Requests Found

Return values

0	Success
Other	Error Code indicating the nature of the problem

Note

> Requests which have already been aborted receive a new abort signal when this API is called. Requests that have been previously aborted but have not completed their abort processing are counted as both 'already aborted' and 'aborted'. To calculate the number of newly aborted requests, do: RequestsAborted - Requests-AlreadyAborted.

9.2.6.14 int hpss_AcctCodeToName (acct_rec_t *AcctCode,* hpss_id_t ∗ *Site,* char ∗ *AcctName*)

Maps an account code to its corresponding account name.

The 'hpss_AcctCodeToName' function maps an Account Code to the corresponding Account Name. If the site is not specified, then the default site is used.

Parameters

in	*AcctCode*	Account Code to look up
in,out	*Site*	UUID of the Site; if NULL the default site for the current working directory is used
out	*AcctName*	Account Name associated with the given Account Code. Account Name should be a string of at least HPSS_MAX_ACCOUNT_NAME characters.

Return values

0	No error. Account Code found and returned.
-EFAULT	Account Name is a NULL pointer.
-EINVAL	The given Account Code is not valid.

See Also

hpss_AcctNameToCode

9.2.6.15 int hpss_AcctNameToCode (char * *AcctName,* hpss_id_t * *Site,* acct_rec_t * *AcctCode*)

Maps an account name to its corresponding account code.

The 'hpss_AcctNameToCode' function maps an account name to the corresponding account code. If the site is not specified, then the default site will be used.

Parameters

`in,out`	*AcctName*	Account Name to look up
`in,out`	*Site*	UUID of the site; if NULL then the default site is used.
`out`	*AcctCode*	Account Code associated with the given Account Name

Return values

0	No error. Account Code found and returned.
-EFAULT	Account Name or Account Code is NULL.
-EINVAL	The given Account Name is not valid.
-ENAMETOOLONG	The Account Name provided has too many characters.

See Also

hpss_AcctCodeToName

9.2.6.16 char* hpss_BuildLevelString (void)

Construct a string containing HPSS version information.

Retrieve the HPSS build level string in Major.Minor.Maintenance.Patch.Efix format.

Returns

A string containing the HPSS version information in MAJOR.MINOR.MAINTENANCE.PATCH.EFIX format, e.g. "7.4.1.0.0". NULL is returned if an error allocating memory occurs.

Return values

ENOMEM	Insufficient Memory

Note

This version string corresponds to using the 5 levels specified in the hpss_version.h header file.
The macro HPSS_LEVEL can also be used to obtain a (numeric) representation of the HPSS version.
The caller is responsible for freeing the returned string.

9.2.6.17 void hpss_ClearLastHPSSErrno (void)

Clears the current HPSS errno state.

Zeros out all of the information in the thread-specific error context.

9.2.6.18 int hpss_ConvertIdsToNames (int32_t *NumEntries*, api_namespec_t ∗ *Specs*)

Converts user or group ids to names.

This function converts one or more ids into their corresponding names.

Parameters

`in`	*NumEntries*	# entries in Specs
`in,out`	*Specs*	Principals to convert

Return values

0	All ids were translated successfully
-EINVAL	The Specs array has an entry that contains a zero realm id.
-ENOCONNECT	A problem with the security registry.
-EFAULT	A name is too long to fit into Specs.
-EAGAIN	Default for security registry error.

Note

The *Specs* must contain *NumEntries* elements of type api_name_spec_t. Each array element has a *Type* field that determines how the element will be translated. If the *Type* is NAMESPEC_SKIP, then the *Id* and *RealmId* fields will be ignored, and the *Name* and *RealmName* fields will be set to undefined values. NAMESPEC_-SKIP enables ACL editors to deal with ACL entry types, such as the mask object, which do not contain any ids.
If the Type is set to NAMESPEC_REALM, then the RealmId field will be translated into a realm name, which is returned in the RealmName field. In this case, the Id field will be ignored and the Name field will be set to an undefined value. On the other hand, if the Type is set to NAMESPEC_USER or NAMESPEC_GROUP, then the function returns the user or group name in the Name field as well as filling in the RealmName field.
If, at any point during the translation process, an array entry is found that cannot be translated, the routine will return immediately. In this case, the Name and RealmName fields of each Specs array entry should be considered undefined, but the Type, Id and RealmId fields will remain unchanged from their initial values. Therefore the caller may need to keep a copy of the Specs array if the name fields will be needed again.
If the principal cannot be found in the desired realm, then the routine does not return an error, but rather just stores an ASCII representation of the principal's id in the Name field. Likewise, if the realm cannot be found in the trusted realm table, then the routine returns the realm id in the RealmName field. This allows the caller to deal with principals and/or realms that no longer exist. The caller may detect this “error” by scanning for numeric data in these fields.

9.2.6.19 int hpss_ConvertNamesToIds (int32_t *NumEntries*, api_namespec_t ∗ *Specs*)

Converts user or group names to ids.

This function converts one or more names into the corresponding ids.

Parameters

`in`	*NumEntries*	Number of entries in Specs
`in,out`	*Specs*	Specs to convert

Return values

0	All names were translated successfully
-EFAULT	No namespec provided
-ENOENT	One or more entries in the Specs array specified a principal or realm name that could not be found.
-ENOCONNECT	A problem with the security registry.
-EINVAL	Non-numeric principal id specified
-EINVAL	One or more spec types provided were invalid

Note

The Specs array must contain NumEntries elements of type api_name_spec_t. Each array entry has a Type field that determines how that element will be translated. If the element's Type is NAMESPEC_SKIP, then the Name and RealmName fields will be ignored and the Id and RealmId fields will be set to undefined values. This is used to help ACL editors deal with ACL entry types, such as the mask object, which do not contain any names.
If the element's Type is set to NAMESPEC_REALM, the RealmName field is translated into a realm id, which is returned in the RealmId field. In this case, the Name field will be ignored and the Id field will be set to an undefined value. On the other hand, if Type is set to NAMESPEC_USER or NAMESPEC_GROUP, the function returns the UID or GID in the Id field as well as filling in the RealmId field. If the element's Name and/or RealmName fields are strings or digits, the routine does not try to verify that the principal and/or realm actually exist. Rather, it just returns the corresponding values in the Id and RealmId fields. This allows the caller to deal with principals and/or realms that no longer exist. In this situation, the routine does not return ENOENT.
If at any point during the translation process, an array entry is found that cannot be translated, the routine will return immediately. In this case, the Id and RealmId fields of each Specs array entry should be considered undefined, but the Type, Name, and RealmName fields will remain unchanged from their initial values. Therefore the caller may need to keep a copy of the Specs array if the id fields will be needed again.

9.2.6.20 char** hpss_CopyrightStrings (void)

Read the copyright file and return the strings.

Returns

An array of strings read from the copyright file

9.2.6.21 int hpss_FdelFileDigestList (int *Fildes,* hpss_file_hash_stripe_flags_t *Flags*)

Delete file hash digest information.

Delete file digest information. This is a wrapper around the set API to facilitate deleting file digests. This API takes a striping preference.

If the preference is not only single or only striped, both types of digests are attempted to be deleted. Both hash types will be attempted to be deleted before anything is returned. The function will return failure if either type fails to be deleted.

If the preference is only for single or striped, only that digest will be deleted.

Parameters

`in`	*Fildes*	Open bitfile descriptor
`in`	*Flags*	Digest deletion preference

Return values

0	Success
-EBADF	Fildes is not a valid open descriptor
-EINVAL	Hash type is invalid
-EBUSY	Fildes is a busy descriptor
-ESTALE	Fildes is a stale descriptor and should be reopened
-EFAULT	Buffer is an invalid pointer

9.2.6.22 int hpss_FgetFileDigest (int *Fildes,* hpss_file_hash_digest_t ∗ *Digest*)

Retrieves the file's hash digest, if any, for an open file.

The 'hpss_FgetFileDigest' function returns the file digest information. The caller allocates a buffer to hold the returned hash digest information and passes the pointer in the 'Digest' parameter. On successful return the supplied buffer is populated with the digest information.

Parameters

`in`	*Fildes*	Open bitfile descriptor
`in,out`	∗*Digest*	Returned hash digest information

Return values

0	Success
-EBADF	Fildes is not a valid open descriptor
-ENOENT	Digest information doesn't exist
-EBUSY	Fildes is a busy descriptor
-ESTALE	Fildes is a stale descriptor and should be reopened
-EFAULT	Digest output parameter is invalid

Note

The output buffer will contain the binary value for the hash digest.
The 'Flags' field of the digest information is used to indicate if a hash value is returned. The presence of the HPSS_FILE_HASH_DIGEST_VALID flag indicates that the digest buffer contains a valid hash value.

See Also

hpss_FileGetDigestHandle, hpss_FgetFileDigestList

9.2.6.23 int hpss_FgetFileDigestList (int *Fildes,* hpss_file_hash_stripe_flags_t *Flags,* uint64_t ∗ *StripeLength,* hpss_file_hash_digest_list_t ∗ *DigestList*)

Retrieves a file's hash digest, if any, for an open file.

Returns the file digest information. The caller allocates a buffer to hold the returned hash digest information and passes the pointer in the 'DigestList' parameter. On successful return, the supplied buffer is populated with the digest information.

Each file may have a one single stripe checksum and a client stripe (also called a multi stripe) checksum. A multi stripe checksum contains hash data for multiple *client* stripes. Multi stripe hash data can only be validated by reading the file back with the equivalent striping – StripeLength and Stripe Count.

The Flags parameter allows the caller to indicate a preference for which checksum to return.

Parameters

`in`	*Fildes*	Open bitfile descriptor
`in`	*Flags*	Stripe preference flag
`out`	*StripeLength*	Digest stripe length
`out`	*DigestList*	Returned hash digest information

Return values

0	Success
-EBADF	Fildes is not a valid open descriptor
-ENOENT	Digest information doesn't exist
-EBUSY	Fildes is a busy descriptor
-ESTALE	Fildes is a stale descriptor and should be reopened
-EFAULT	DigestList output parameter is invalid

Note

The output buffer will contain the binary value for the hash digest.
The 'Flags' field of the digest information is used to indicate if a hash value is returned. The presence of the HPSS_FILE_HASH_DIGEST_VALID flag indicates that the digest buffer contains a valid hash value.
For multi stripe checksums, the state of the hash can be determined by the presence of the HPSS_FILE_HA-SH_DIGEST_ACTIVE flag, which is set or cleared by the client. For single stripe checksums, this flag is the sameas the HPSS_FILE_HASH_DIGEST_VALID flag since the system manages the hash validity state.

See Also

hpss_GetFileDigestHandle, hpss_FgetFileDigest

9.2.6.24 int hpss_FileGetDigestHandle (const ns_ObjHandle_t ∗ *ObjHandle*, const char ∗ *Path*, sec_cred_t ∗ *Ucred*, hpss_file_hash_digest_t ∗ *Digest*)

Retrieves the file's hash digest, if any.

The 'hpss_FileGetDigestHandle' function returns the file digest information. The caller allocates a buffer to hold the returned hash digest information and passes the pointer in the 'Digest' parameter. On successful return the supplied buffer is populated with the digest information.

Parameters

`in`	∗*ObjHandle*	parent object handle
`in`	∗*Path*	path of file to get attributes
`in`	∗*Ucred*	user credentials
`in,out`	∗*Digest*	Returned hash digest information

Return values

0	Success
-ENOENT	File doesn't exist
-EINVAL	One or more of the parameters are invalid
-EFAULT	Digest output parameter is invalid

Note

The output buffer will contain the binary value for the hash digest.
A hash type of 'hpss_hash_type_none' will indicated the absence of a hash record.
The 'Flags' field of the digest information is used to indicate if a hash value is returned. The presence of the HPSS_FILE_HASH_DIGEST_VALID flag indicates that the digest buffer contains a valid hash value.

See Also

hpss_FgetFileDigest, hpss_FileGetDigestHandle

9.2.6.25 int hpss_FileGetDigestListHandle (const **ns_ObjHandle_t** ∗ *ObjHandle,* const char ∗ *Path,* sec_cred_t ∗ *Ucred,* **hpss_file_hash_stripe_flags_t** *Flags,* uint64_t ∗ *StripeLength,* **hpss_file_hash_digest_list_t** ∗ *DigestList*)

Retrieves a file's hash digest(s) by preference, if any.

Returns file digest information. The caller provides a buffer to hold the returned hash digest information, and passes the pointer in the 'DigestList' parameter. On successful return the supplied buffer is populated with the digest information. The caller is responsible for freeing the digest value array, which is allocated by this function.

Parameters

`in`	∗*ObjHandle*	parent object handle
`in`	∗*Path*	path of file to get attributes
`in`	∗*Ucred*	user credentials
`in`	*Flags*	Stripe preferencing
`out`	∗*StripeLength*	Length of each stripe, in bytes • If the digest is not striped this value is unused.
`out`	∗*DigestList*	Returned hash digest information

Return values

0	Success
-ENOENT	File doesn't exist
-EINVAL	One or more of the parameters are invalid
-EFAULT	Digest output parameter is invalid

Note

The output buffer will contain the binary value for the hash digest.
A hash type of 'hpss_hash_type_none' will indicated the absence of a hash record.
The 'Flags' field of the digest information is used to indicate if a hash value is returned. The presence of the HPSS_FILE_HASH_DIGEST_VALID flag indicates that the digest buffer contains a valid hash value.

See Also

hpss_FgetFileDigest, hpss_FileGetDigestHandle

9.2.6.26 int hpss_FsetFileDigest (int *Fildes,* const hpss_file_hash_digest_t ∗ *Digest*)

Sets single stripe file's hash digest information.

The 'hpss_FsetFileDigest' function sets the file digest information. The 'Type' digest field specifies one of the valid hash types, or none to drop the hash information.

Parameters

`in`	*Fildes*	Open bitfile descriptor
`in`	*Digest*	Digest information pointer

Return values

0	Success
-EBADF	Fildes is not a valid open descriptor
-EINVAL	Hash type is invalid
-EBUSY	Fildes is a busy descriptor
-ESTALE	Fildes is a stale descriptor and should be reopened
-EFAULT	Buffer is an invalid pointer

Note

The input buffer should contain the binary value for the hash digest.
The 'Flags' field of the digest information is used to indicate that a hash value is to be set. The presence of the HPSS_FILE_HASH_DIGEST_VALID flag indicates that the digest buffer contains a valid hash value.

9.2.6.27 int hpss_FsetFileDigestList (int *Fildes,* uint64_t *StripeLength,* const hpss_file_hash_digest_list_t ∗ *DigestList*)

Sets file hash digest information.

Sets file digest information. If the digest list specifies a single digest, the digest is set just as it would be in hpss_-FsetFileDigest. If the digest list specifies more than one digest, they are considered to be for a *client striped digest* , or multi stripe digests. A client striped digest does not function as an "file digest" in the End to End Data Integriy (E2EDI) feature; the system does not interact with it and it is up to the client to maintain or validate the striped checksums.

For multi stripe digests, this API behaves similarly to single stripe. Each digest should contain identical information, aside from the buffer data. The order of the digests in the list provided will be persisted for retrievals.

If the HPSS_FILE_HASH_DIGEST_VALID flag is not provided, the digest information will be deleted. Multi stripe checksums can persist the validity of the stored checksum by passing in the HPSS_FILE_HASH_DIGEST_ACTIVE flag. This flag has no effect upon single stripe digests, as the system maintains the validity state of those digests.

Parameters

`in`	*Fildes*	Open bitfile descriptor
`in`	*StripeLength*	Stripe Length for digests
`in`	*DigestList*	Digest information pointer

Return values

0	Success
-EBADF	Fildes is not a valid open descriptor
-EINVAL	Hash type is invalid
-EBUSY	Fildes is a busy descriptor
-ESTALE	Fildes is a stale descriptor and should be reopened
-EFAULT	Buffer is an invalid pointer

Note

> The StripeLength parameter is unused in the single stripe case.
> The input buffer should contain the binary value for the hash digest.
> The 'Flags' field of the digest information is used to indicate that a hash value is to be set. The presence of the HPSS_FILE_HASH_DIGEST_VALID flag indicates that the digest buffer contains a valid hash value.

9.2.6.28 int hpss_GetAcct (acct_rec_t ∗ *RetDefAcct*, acct_rec_t ∗ *RetCurAcct*)

Retrieves current and default account codes.

The 'hpss_GetAcct' function gets the default and current account codes for the calling thread.

Parameters

`out`	*RetDefAcct*	default account code
`out`	*RetCurAcct*	current account code

Return values

0	Retrieved codes successfully
-EFAULT	Either RetDefAcct or RetCurAcct is invalid
<0	Error from API_ClientAPIInit()

9.2.6.29 int hpss_GetAcctName (char ∗ *AcctName*)

Retrieves account name.

The 'hpss_GetAcctName' function returns the current account name for this thread. Since each site contacted by each thread in the Client API can have its own session account name, the account name for the site managing the current working directory is returned.

Parameters

`out`	*AcctName*	Current account name; should be at least HPSS_MAX_ACCOUNT_NAME characters long

Return values

0	Success
<0	Error from API_ClientAPIInit()
-EFAULT	A NULL AcctName was provided.

9.2.6.30 int hpss_GetAllSubsystems (ls_map_array_t ∗∗ *ls_map_array*)

Retrieve a map of all subsystems.

Parameters

`out`	*ls_map_array*	Location Map Array

This function finds all subsystems for the current site.

Return values

0	Success
HPSS_ENOENT	No subsystems were returned.
Other	Error communicating with the location server

Note

The caller is responsible for freeing the returned map.

9.2.6.31 int hpss_GetAsyncStatus (**hpss_reqid_t** *CallBackId,* **bfs_bitfile_obj_handle_t** ∗ *BitfileObj,* int32_t ∗ *Status*)

Get the status of a background stage.

This request can be used to get status information on a asynchronous request that was started with callback information provided.

Parameters

`in`	*CallBackId*	callback request identifier
`in`	*BitfileObj*	Bitfile Object
`out`	*Status*	status HPSS_STAGE_STATUS_UNKNOWN - No status available HPSS_ST-AGE_STATUS_QUEUED - Request still in queue and not yet active HPSS_-STAGE_STATUS_ACTIVE - Stage active and in progress

Return values

0	Status has been obtained
-EINVAL	The Bitfile Object parameter is not a valid pointer.
-EFAULT	The Status parameter is a NULL pointer.

9.2.6.32 int hpss_GetBatchAsynchStatus (**hpss_reqid_t** *CallBackId,* **hpss_stage_bitfile_list_t** ∗ *BFObjs,* **hpss_stage_status_type_t** *Type,* **hpss_stage_batch_status_t** ∗ *Status*)

Check the status of files in a batch request.

The hpss_GetBatchAsynchStatus function will provide the status (Not Found, Queued, or Active) for each bitfile specified with the callback id.

Parameters

`in`	*CallBackId*	Callback Id
`in`	*BFObjs*	Bitfile Objects to Status
`in`	*Type*	Status Request Type. Valid types are: • HPSS_BATCH_STATUS_BFS - Only get BFS Status • HPSS_BATCH_STATUS_BFS_SS - Get BFS and SS Status
`out`	*Status*	Batch Status

Return values

0	Status successfully retrieved
-EFAULT	No Bitfile ID list provided
-EFAULT	No Status structure provided
-EINVAL	Invalid Bitfile ID list
Other	Status could not be retrieved

Note

Statuses returned include IO_RUNNING, IO_WAITING, and IO_NOEXIST.

9.2.6.33 int hpss_GetDistFile (int *Fildes*, api_dist_file_info_t ∗ *FileInfo*)

Extracts a file table entry for an open file descriptor.

The 'hpss_GetDistFile' function extracts a file table entry for a supplied open file descriptor. The function returns a pointer to the file table information.

Parameters

in	*Fildes*	Open file handle
out	*FileInfo*	File table information

Return values

0	Success
-EBADF	Invalid file descriptor Fildes.
-EBUSY	Another thread is currently manipulating this entry.
-EFAULT	FileInfo is a NULL pointer.

9.2.6.34 int hpss_GetFullPath (const ns_ObjHandle_t ∗ *ObjHandle*, char ∗∗ *FullPath*)

Provides the full path back to an object from the root of roots.

Returns a path back to the object handle specified.

Parameters

in	*ObjHandle*	Object Handle
out	*FullPath*	Path to object

Return values

0	Success
-EINVAL	Null handle provided
-ENOENT	The path could not be retrieved
-EFAULT	No FullPath buffer passed in

Note

The caller is responsible for freeing the *FullPath*.

9.2.6.35 int hpss_GetFullPathBitfile (const bfs_bitfile_obj_handle_t ∗ *BitfileObj*, char ∗∗ *FullPath*)

Provides a full path back to a bitfile from the root of roots.

Returns a path back to the bitfile specified.

Parameters

in	*BitfileObj*	Bitfile Object Handle
out	*FullPath*	Path to object

Return values

0	Success
-EINVAL	Null handle provided
-ENOENT	The path could not be retrieved
-EFAULT	No FullPath buffer passed in

Note

The caller is responsible for freeing the *FullPath*.

9.2.6.36 int hpss_GetFullPathBitfileLen (const bfs_bitfile_obj_handle_t * *BitfileObj*, char * *Path*, size_t *BufLen*)

Provides a full path back to a bitfile from the root of roots.

Returns a path back to the bitfile specified.

Parameters

in	*BitfileObj*	Bitfile Object Handle
in,out	*Path*	Path to object
in	*BufLen*	Size of *Path*

Return values

0	Success
-ERANGE	Full path is longer than BufLen
-EINVAL	Null handle provided
-ENOENT	The path could not be retrieved
-EFAULT	No FullPath buffer passed in

Note

Unlike hpss_GetFullPathBitfile, this takes a buffer from the caller.

9.2.6.37 int hpss_GetFullPathHandle (const ns_ObjHandle_t * *ObjHandle*, const char * *Path*, char ** *FullPath*)

Provides the full path back to an object from the root of roots.

Returns a path back to the object handle specified.

Parameters

in	*ObjHandle*	Object Handle
in	*Path*	Additional Path
out	*FullPath*	Path to object

Return values

0	Success
-EINVAL	Null handle provided
-ENOENT	The path could not be retrieved
-EFAULT	No FullPath buffer passed in

Note

The caller is responsible for freeing the *FullPath*.

9.2.6.38 int hpss_GetFullPathHandleLen (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *ObjPath,* char ∗ *Path,* size_t *BufLen*)

Provides the full path back to an object from the root of roots.

Returns a path back to the object handle specified.

Parameters

`in`	*ObjHandle*	Object Handle
`in`	*ObjPath*	Additional Path
`in,out`	*Path*	Path to object
`in`	*BufLen*	Size of *Path*

Return values

0	Success
-ERANGE	Path is longer than BufLen
-EINVAL	Null handle provided
-ENOENT	The path could not be retrieved
-EFAULT	No FullPath buffer passed in

Note

Unlike hpss_GetFullPathHandle, this takes a buffer from the caller.

9.2.6.39 int hpss_GetFullPathLen (const ns_ObjHandle_t ∗ *ObjHandle,* char ∗ *Path,* size_t *BufLen*)

Provides the full path back to an object from the root of roots.

Returns a path back to the object handle specified.

Parameters

`in`	*ObjHandle*	Object Handle
`in,out`	*Path*	Path to object
`in`	*BufLen*	Size of *Path*

Return values

0	Success
-ERANGE	Path is longer than BufLen
-EINVAL	Null handle provided
-ENOENT	The path could not be retrieved
-EFAULT	No FullPath buffer passed in

Note

Unlike hpss_GetFullPath, this takes a buffer from the caller.

9.2.6.40 int32_t hpss_GetGID (gid_t ∗ *GID*)

Get the group id for the calling thread.

The 'hpss_GetGID' function gets the current GID for the calling thread.

Return values

0	Success
<0	Error from API_ClientAPIInit().

9.2.6.41 hpss_errno_state_t hpss_GetLastHPSSErrno (void)

Retrieves the error state of the last HPSS error.

Returns

This function returns the last HPSS error state.

Note

The error state is not cleared after it is retrieved.

9.2.6.42 hpss_reqid_t hpss_GetNextIORequestID ( )

Retrieves the next request id that will be used for I/O.

Retrieve the request id that will be used by this thread for its next I/O operation. This includes read, write, stage, migrate, and copy.

For the 'stream' APIs (e.g. hpss_Fwrite()), the next request id holds for the the next stream API call that results in I/O back to HPSS, but the stream API may issue multiple I/O requests.

Return values

This	function returns the next request id to be used for I/O.

Note

For PIO, use hpss_PIOGetRequestID to get the request id for the group.

9.2.6.43 void hpss_HashFlagsToString (unsigned16 *flags*, char ∗ *buf*, int *len*)

Returns a string representation of the provided hash flags.

Parameters

`in`	*flags*	Hash flags
`in,out`	*buf*	String buffer
`in`	*len*	String buffer length

Returns

char ∗

- String representation of the hash flags

9.2.6.44 int hpss_InsertDistFile (const api_dist_file_info_t ∗ *FileInfo*)

Inserts a file table entry into the current file table.

The 'hpss_InsertDistFile' function attempts to insert an extracted file table entry into the current file table.

Parameters

`in`	*FileInfo*	File table information

Return values

0	Success
-EFAULT	FileInfo is a NULL pointer.
-EINVAL	Checksum on file information failed.
-EMFILE	Too many open file descriptors.
-EIO	An input/output or HPSS internal error occurred.

Note

A file entry can only be inserted if it a had previously been extracted using hpss_GetDistFile.

9.2.6.45 int hpss_LoadThreadState (uid_t *UserID,* mode_t *Umask,* const char ∗ *ClientFullName*)

Allows some clients to manipulate the local state of a thread.

Special interface to allow well-behaved clients to manipulate the thread's local state.

Parameters

UserID	New user credentials
Umask	New umask
ClientFullName	Client Fully Qualified Name

Return values

0	No error. Thread specific state is valid.
-EINVAL	NumGroups is invalid
-ENOENT	Credentials for the specified user could not be obtained.
-ENOMEM	Unable to allocate memory

9.2.6.46 int hpss_LookupRootCS (const char ∗ *SiteName,* hpss_srvr_id_t ∗ *ServerId*)

Retrieves the ID for a site's root Core Server.

This routine returns the ID for the root Core Server specified by the provided HPSS site name.

Parameters

in	*SiteName*	HPSS site name
out	*ServerId*	Root Core Server server ID

Return values

-EINVAL	The SiteName parameter is invalid
-EFAULT	The ServerId parameter is NULL
-ENOENT	The Root Core Server could not be located

9.2.6.47 int hpss_Migrate (int *Fildes,* uint32_t *SrcLevel,* uint32_t *Flags,* uint64_t ∗ *RetBytesMigrated*)

Migrates data in an open file from a specified hierarchy level.

The ’hpss_Migrate’ function requests that data in the open file be migrated from the specified level in the storage hierarchy.

Parameters

in	*Fildes*	ID of open file/directory
in	*SrcLevel*	from where to migrate data
in	*Flags*	controlling flags
out	*RetBytes-Migrated*	number of bytes migrated

Return values

0	Success
-EBADF	The supplied file descriptor does not correspond to a file opened for writing.
-EBUSY	The specified file descriptor is in use.
-EFAULT	The RetBytesMigrated parameter is a NULL pointer.
-EINVAL	The Flags argument is invalid.
-EPERM	The client does not have the appropriate privileges to issue explicit file migration requests.

9.2.6.48 int hpss_PingCore (const **hpss_srvr_id_t** ∗ *CoreServerID,* uint32_t ∗ *RecvSecs,* uint32_t ∗ *RecvUSecs*)

Pings the Core Server.

This routine is called to issue a ping to the Core Server. The seconds and microseconds representing the time (relative to the Unix Epoch) the request is received at the server is returned.

Parameters

in	*CoreServerID*	Core Server ID
out	*RecvSecs*	Secs received
out	*RecvUSecs*	USecs received

Return values

0	Success
-EFAULT	Null parameter
-ENOCONNECT	Connection to the Core Server could not be established.

Note

This routine is used to test and verify network performance between the client and the HPSS core components.

9.2.6.49 int hpss_Purge (int *Fildes,* uint64_t *Offset,* uint64_t *Length,* uint32_t *StorageLevel,* uint32_t *Flags,* uint64_t ∗ *RetBytesPurged*)

Purges data in an open file from a specified hierarchy level.

The 'hpss_Purge' function requests that data in the open file be purged from the specified level in the storage hierarchy.

Parameters

in	*Fildes*	ID of open file/directory
in	*Offset*	beginning offset
in	*Length*	length to be purged
in	*StorageLevel*	where to purge data
in	*Flags*	controlling flags
out	*RetBytesPurged*	number of bytes purged

Return values

0	No error.
-EBADF	The supplied file descriptor does not correspond to an open file.
-EBUSY	The specified file descriptor is in use.
-EFAULT	The RetBytesPurged parameter is a NULL pointer.
-EINVAL	The Flags, Offset or Length argument is invalid.
-EPERM	The client does not have the appropriate privileges to perform explicit purge operations.

9.2.6.50 int hpss_SetAcct (acct_rec_t *NewCurAcct*)

Sets the current account code.

The 'hpss_SetAcct' function sets the current account code for the calling thread.

Parameters

in	*NewCurAcct*	new value for current code

Return values

0	success
-EFAULT	The account name with this account code is NULL.
-EINVAL	The client is configured for Unix-style accounting, and therefore the account code cannot be changed.
-ENOENT	Account doesn't exist.
-ENOPERM	User is not a member of this account.
<0	Error from API_ClientAPIInit()

9.2.6.51 int hpss_SetAcctByName (const char ∗ *NewAcctName*)

Sets the current account name.

The 'hpss_SetAcctByName' function sets the current account information based on the NewAcctName for the calling thread.

Parameters

`in`	*NewAcctName*	new value for current account

Return values

0	Success
-EFAULT	The account name provided is NULL.
-ENAMETOOLONG	The account name provided contains too many characters.
-ENOENT	The account specified by NewAcctName doesn't exist.
-EPERM	The user does not have sufficient privilege to change the account name to NewAcctName, or NewAcctName is not defined at the site indicated by the client's working directory.
<0	Error from API_ClientAPIInit()

9.2.6.52 int hpss_SetGID (uint32_t *NewCurGid*)

Set the group id for the calling thread.

The 'hpss_SetGID' function sets the current GID for the calling thread, and the current GID in the global threadstate.

Parameters

`in`	*NewCurGid*	new value for current GID

Return values

0	Success
-EINVAL	The NewGroupID parameter is invalid.
<0	Error from API_ClientAPIInit().

9.2.6.53 int hpss_SiteIdToName (const hpss_id_t ∗ *SiteId,* char ∗ *SiteName*)

Convert Site ID to Site Name.

This function converts one site id to the corresponding site name.

Parameters

`in`	*SiteId*	site id
`out`	*SiteName*	site name

Return values

0	SiteId was translated successfully.
-ENOENT	The value passed in SiteId does not correspond to a known HPSS site.
-ECONN	There was a communication problem during the translation.

9.2.6.54 int hpss_SiteNameToId (const char ∗ *SiteName,* hpss_id_t ∗ *SiteId*)

Convert Site Name to Site ID.

This function converts one site name to it's corresponding id.

Parameters

in	*SiteName*	site name
out	*SiteId*	site id

Return values

0	SiteName was translated successfully.
-ENOENT	The value passed in SiteName does not correspond to a known HPSS site.
-ECONN	There was a communication problem during the translation.

9.2.6.55 int hpss_Stage (int *Fildes,* uint64_t *Offset,* uint64_t *Length,* uint32_t *StorageLevel,* uint32_t *Flags*)

Stage an open HPSS file.

The 'hpss_Stage' function requests that data in the open file be staged to the highest level in the storage hierarchy.

Parameters

in	*Fildes*	ID of open file/directory
in	*Offset*	beginning offset
in	*Length*	length to be staged
in	*StorageLevel*	where to stage data
in	*Flags*	controlling flags

Return values

0	Success
-EBADF	The supplied file descriptor does not correspond to an open file.
-EINVAL	The Flags, Offset or Length argument is invalid.
-EBUSY	The specified file descriptor is currently in use.
-EPERM	The client does not have the appropriate privileges to perform the operation.
-EREMOTE	The caller supplied the BFS_NO_TAPE stage flag, and some part of the file would be staged from tape.

Note

Valid flags:

- BFS_STAGE_ALL (Stage All Data)
- BFS_NO_FAR (Disable Full Aggregate Recall)
- BFS_NO_TAPE (Do not stage from tape)

9.2.6.56 int hpss_StageBatch (hpss_stage_batch_t * *Batch,* hpss_stage_batch_status_t * *Status*)

Stage a batch of bitfiles.

Parameters

in	*Batch*	Batch Stage Information
out	*Status*	Batch Stage Status

The hpss_StageBatch function will create a batch stage request for a group of files specified by Batch. Each of the bitfiles specified will be attempted to be staged, and the status of each will be reflected in the Status parameter.

Return values

0	Requests successfully batched
-EFAULT	NULL Batch and/or Status Structure Provided
-EBADF	One or more file descriptors in the batch stage structure was invalid
-EBUSY	One or more of the file descriptors in the batch stage structure was busy
-ESTALE	One or more of the file descriptors in the batch stage structure is no longer valid
-ERANGE	The size of the batch is larger than the legal size. Split the batch into multiple batches of HPSS_MAX_STAGE_BATCH_LEN or smaller and try again.
-EREMOTE	The caller supplied the BFS_NO_TAPE stage flag, and some part of the file would be staged from tape.
Other	One or more requests failed to stage; review the Status structure for more information.

9.2.6.57 int hpss_StageBatchCallBack (hpss_stage_batch_t ∗ *Batch*, bfs_callback_addr_t ∗ *CallBackPtr*, hpss_reqid_t ∗ *ReqID*, hpss_stage_bitfile_list_t ∗ *BFObjsList*, hpss_stage_batch_status_t ∗ *Status*)

Stage a batch of bitfiles asynchronously.

Parameters

`in`	*Batch*	Batch Stage Information
`in`	*CallBackPtr*	Callback Pointer
`out`	*ReqID*	Request Id
`out`	*BFObjsList*	Stage Bitfile Objects
`out`	*Status*	Status of batch queued entries

The hpss_StageBatchCallback function will create a batch stage request for an array of files specified by Batch. Each of the bitfiles specified will be attempted to be staged separately. The request id will be provided back to the application so that it may check the status of the background requests. It will also return a list of bitfile ids associated with the stage requests.

Return values

0	Requests successfully batched
-EFAULT	NULL Request, Batch, Status or Bitfile ID Structure Provided
-EINVAL	Invalid Batch Structure Length Provided
-ERANGE	The size of the batch is larger than the legal size. Split the batch into multiple batches of HPSS_MAX_STAGE_BATCH_LEN or smaller and try again.
Other	One or more requests failed to stage; review the Status structure for more information.

9.2.6.58 int hpss_StageCallBack (const char ∗ *Path*, uint64_t *Offset*, uint64_t *Length*, uint32_t *StorageLevel*, bfs_callback_addr_t ∗ *CallBackPtr*, uint32_t *Flags*, hpss_reqid_t ∗ *ReqID*, bfs_bitfile_obj_handle_t ∗ *BitfileObj*)

Stage an HPSS file in the background.

This function can be used to start a stage in the background without the file being open. Callback paramaters can be provided to use as setup for later notification.

Parameters

`in`	*Path*	path to the object

in	*Offset*	beginning offset
in	*Length*	length to be staged
in	*StorageLevel*	where to stage data
in,out	*CallBackPtr*	callback information
in	*Flags*	controlling flags
out	*ReqID*	assigned request code
out	*BitfileObj*	needed for later status

Return values

0	No error, caller has access and stage has been started.
-EACCESS	Search permission is denied for a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	The value of the Offset parameter is beyond the end of the file or the StorageLevel parameter is invalid for the storage hierarchy.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the pathname exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

Note

If no id is specified in the callback, this function places the generated request id in as the callback id.
Request ids should be generated by API_GetUniqueRequestID.
Valid flags:

- BFS_STAGE_ALL (Stage All Data)
- BFS_NO_FAR (Disable Full Aggregate Recall)
- BFS_NO_TAPE (Do not stage from tape)

9.2.6.59 int hpss_StageCallBackBitfile (const **bfs_bitfile_obj_handle_t** ∗ *BitfileObj*, uint64_t *Offset*, uint64_t *Length*, uint32_t *StorageLevel*, **bfs_callback_addr_t** ∗ *CallBackPtr*, uint32_t *Flags*, **hpss_reqid_t** ∗ *ReqID*)

Stage a file in the background using its bitfile id.

This function can be used to start a stage in the background without the file being open. Callback paramaters can be provided to use as setup for later notification.

Parameters

in	*BitfileObj*	bitfile object
in	*Offset*	beginning offset
in	*Length*	length to be staged
in	*StorageLevel*	where to stage data
in,out	*CallBackPtr*	callback info
in	*Flags*	controlling flags
out	*ReqID*	assigned request code

Return values

0	No error, caller has access and stage has been started.

Note

If no id is specified in the callback, this function places the generated request id in as the callback id.
Request ids should be generated by API_GetUniqueRequestID.
Valid flags:

- BFS_STAGE_ALL (Stage All Data)
- BFS_NO_FAR (Disable Full Aggregate Recall)
- BFS_NO_TAPE (Do not stage from tape)

9.2.6.60 mode_t hpss_Umask (mode_t *CMask*)

Set the file mode creation mask, and returns the previous mask value.

The 'hpss_Umask' function sets the file mode creation mask of the thread and returns the previous value of the mask.

Parameters

`in`	*CMask*	New file creation mask

Returns

Previous value of the mask.

9.2.6.61 int hpss_UserAttrListAttrHandle (const ns_ObjHandle_t ∗ *Obj,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* hpss_userattr_list_t ∗ *Attrs,* int *Flags,* int *XMLSize*)

List User-defined Attributes associated with a namespace path using a handle.

Parameters

`in`	*Obj*	Parent Object
`in`	*Path*	Path
`in`	*Ucred*	user credentials
`out`	*Attrs*	Attributes
`in`	*Flags*	Flags
`in`	*XMLSize*	XML Size

Obtain all attributes associated with Handle and Path. Additionally, this function will put occurrence identifiers next to attributes with multiple occurrences, and obtain XML element attributes. The results are guaranteed to be in the order they appear in the XML document.

Return values

-EACCES	The user does not have permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer and Obj is NULL.
-EINVAL	The XML from Path could not be converted into Attribute format.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.
-EACCES	The user does not have write permission on the namepsace object, or search permission is denied on a component of the path prefix.
-ERANGE	The length of the data requested was too large to be returned.
-EDOM	The XMLSize was too large.

Warning

The attributes returned by this function must be freed by the caller.

9.3 File Open, Create, and Close

Functions which can be used to open and close HPSS files.

Functions

- int Common_hpss_UnlinkHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, int Immediate)

  Remove a namespace entry using a handle, bypassing the trashcan.
- int hpss_Close (int Fildes)

  Terminates the connection between a file and its handle.
- int hpss_Creat (const char ∗Path, mode_t Mode, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities-_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

  Create a file and open it.
- int hpss_Create (const char ∗Path, mode_t Mode, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_-priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

  Create an HPSS file.
- int hpss_CreateHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, mode_t Mode, sec_cred_t ∗Ucred, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗Hints-Out, hpss_vattr_t ∗AttrsOut)

  Create a file using a handle.
- int hpss_Open (const char ∗Path, int Oflag, mode_t Mode, const hpss_cos_hints_t ∗HintsIn, const hpss_cos-_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

  Optionally create and open an HPSS file.
- int hpss_OpenBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, int Oflag, const sec_cred_t ∗Ucred)

  Open an HPSS file by bitfile id.
- int hpss_OpenBitfileVAttrs (const hpss_vattr_t ∗FileAttrs, int Oflag, const sec_cred_t ∗Ucred, hpss_cos_-hints_t ∗HintsOut, uint64_t ∗SegmentSize)

  Open a bitfile using vattrs.
- int hpss_OpenHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, int Oflag, mode_t Mode, sec-_cred_t ∗Ucred, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut, hpss_vattr_t ∗AttrsOut)

  Optionally create and open an HPSS file relative to a given directory.
- int hpss_ReopenBitfile (int Fildes, const bfs_bitfile_obj_handle_t ∗BitfileObj, int Oflag, const sec_cred_t ∗Ucred)

  Open an HPSS file by bitfile object handle.
- int hpss_Unlink (const char ∗Path)

  Remove a namespace entry.
- int hpss_UnlinkHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Remove a namespace entry using a handle.
- int hpss_UnlinkHandleImmediate (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Remove a namespace entry using a handle, bypassing the trashcan.
- int hpss_UnlinkImmediate (const char ∗Path)

  Remove a namespace entry, bypassing the trashcan.

9.3.1 Detailed Description

Functions which can be used to open and close HPSS files.

9.3.2 Function Documentation

9.3.2.1 int Common_hpss_UnlinkHandle (const ns_ObjHandle_t * *ObjHandle,* const char * *Path,* const sec_cred_t * *Ucred,* int *Immediate*)

Remove a namespace entry using a handle, bypassing the trashcan.

The 'hpss_UnlinkHandleImmediate' function removes the entry name by 'Path', taken relative to the directory indicated by 'ObjHandle', and decrements the link count for the file. If link count becomes zero, the file will be deleted when it is no longer open to any client.

Parameters

`in`	*ObjHandle*	parent object handle
`in`	*Path*	path of file to get statistics
`in`	*Ucred*	user credentials
`in`	*Immediate*	Bypass the trashcan

Return values

0	Unlink was successful
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	ObjHandle is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The file named by Path is a directory.

Note

This API causes the trashcan to be bypassed. This API should only be used in specific circumstances where the user indicates they want to bypass being able to recover their files.

9.3.2.2 int hpss_Close (int *Fildes*)

Terminates the connection between a file and its handle.

The 'hpss_Close' function terminates the connection between the file handle, *Fildes*, and the file to which it is associated. The file handle and any associated resources are deallocated and can be reused by a subsequent call to 'hpss_Open'.

Parameters

`in`	*Fildes*	ID of object to be closed

Return values

0	No error. The connection is closed.
-EBADF	The specified file descriptor is out of range, or does not refer to an open file.
-EBUSY	The file is currently in use by another client thread.

See Also

hpss_Open hpss_OpenBitfile hpss_ReopenBitfile

9.3.2.3 int hpss_Creat (const char ∗ *Path,* mode_t *Mode,* const **hpss_cos_hints_t** ∗ *HintsIn,* const **hpss_cos_priorities_t** ∗ *HintsPri,* **hpss_cos_hints_t** ∗ *HintsOut*)

Create a file and open it.

The 'hpss_Creat' function creates a file specified by by *Path*, with permissions as specified by *Mode* and using the class of service values specified by *HintsIn* and *HintsPri*, if non-NULL. It opens the file and returns the open file descriptor.

Parameters

`in`	*Path*	Path to file to be opened
`in`	*Mode*	Desired file perms if create
`in`	*HintsIn*	Desired class of service
`in`	*HintsPri*	Priorities of hint struct
`out`	*HintsOut*	Granted class of service

Return values

$>$0	Open file handle
Zero	Success
$<$0	Error indicating the nature of the failure (see hpss_Open).

Note

An open file descriptor is returned, which must be closed using hpss_Close or a similar function.
This is equivalent to calling hpss_Open with the *Oflag* parameter as O_WRONLY | O_CREAT | O_TRUNC.

9.3.2.4 int hpss_Create (const char ∗ *Path,* mode_t *Mode,* const **hpss_cos_hints_t** ∗ *HintsIn,* const **hpss_cos_priorities_t** ∗ *HintsPri,* **hpss_cos_hints_t** ∗ *HintsOut*)

Create an HPSS file.

The 'hpss_Create' function creates a file specified by by 'Path', with permissions as specified by 'Mode' and using the class of service values specified by 'HintsIn' and 'HintsPri', if non-NULL.

Parameters

`in`	*Path*	Path to file to be opened
`in`	*Mode*	Desired file perms if create
`in`	*HintsIn*	Desired class of service
`in`	*HintsPri*	Priorities of hint struct
`out`	*HintsOut*	Granted class of service

Return values

0	Success
-EACCESS	Search permission is denied on a component of the Path prefix or the file does not exist and write permission is denied for the parent directory of the file to be created.
-EAGAIN	Gatekeeper retries have timed out.
-EEXIST	The named file exists.
-EFAULT	Path is NULL.

-EINVAL	One or more values in the HintsIn parameter is invalid.
-ENAMETOOLONG	The length of the Path string exceeds the system -imposed path name limit or a path name component exceeds the system-imposed limit.
-ENOENT	Path is an empty string.
-ENOMEM	Memory could not be allocated for path name.
-ENOTDIR	A component of the Path prefix is not a directory.

9.3.2.5 int hpss_CreateHandle (const **ns_ObjHandle_t** ∗ *ObjHandle,* const char ∗ *Path,* mode_t *Mode,* sec_cred_t ∗ *Ucred,* const **hpss_cos_hints_t** ∗ *HintsIn,* const **hpss_cos_priorities_t** ∗ *HintsPri,* **hpss_cos_hints_t** ∗ *HintsOut,* hpss_vattr_t ∗ *AttrsOut*)

Create a file using a handle.

The 'hpss_Create' function creates a file specified by by *Path*, with permissions as specified by *Mode* and using the class of service values specified by *HintsIn* and *HintsPri*, if non-NULL.

Parameters

`in`	*ObjHandle*	Parent object handle
`in`	*Path*	Path to file to be opened
`in`	*Mode*	Desired file perms if create
`in,out`	*Ucred*	User credentials
`in`	*HintsIn*	Desired class of service
`in`	*HintsPri*	Priorities of hint struct
`out`	*HintsOut*	Granted class of service
`out`	*AttrsOut*	File attributes

Return values

0	Success.
-EACCESS	Search permission is denied on a component of the Path prefix or the file does not exist and write permission is denied for the parent directory of the file to be created.
-EAGAIN	Gatekeeper retries have timed out.
-EEXIST	The named file exists.
-EFAULT	Path is NULL.
-EINVAL	One or more values in the HintsIn parameter is invalid.
-ENAMETOOLONG	The length of the Path string exceeds the system -imposed path name limit or a path name component exceeds the system-imposed limit.
-ENOENT	Path is an empty string.
-ENOSPC	Resources could not allocated for the new file.
-ENOTDIR	A component of the Path prefix is not a directory.

Note

When account validation is done, the supplied credential account code may be changed based upon account validation requirements.

9.3.2.6 int hpss_Open (const char ∗ *Path,* int *Oflag,* mode_t *Mode,* const **hpss_cos_hints_t** ∗ *HintsIn,* const **hpss_cos_priorities_t** ∗ *HintsPri,* **hpss_cos_hints_t** ∗ *HintsOut*)

Optionally create and open an HPSS file.

Parameters

`in`	*Path*	Names the file to be opened or created.
`in`	*Oflag*	Specifies the file status and access modes to be assigned. Applicable values give below below may be OR'd together. Refer to POSIX.1 for specific behavior. In addition the following HPSS flags are allowed: • HPSS_O_NO_TAPE • HPSS_O_EXCL • HPSS_O_STAGE_ASYNC • HPSS_O_STAGE_BKGRD
`in`	*Mode*	Specifies the file mode for a file that is created as a result of O_CREAT.
`in`	*HintsIn*	Provides allocation hints to HPSS as to the expected structure or access of the file. This argument may be a NULL pointer if the default COS is to be used. This parameter is only used during file creation.
`in`	*HintsPri*	Provides the relative priorities associated with the fields contained in the HintsPri structure. This parameter is only used during file creation, and may be a NULL pointer. The priority will be set to REQUIRED_PRIORITY if HintsPri is NULL.
`out`	*HintsOut*	Contains the hint values actually used when the file is created. This argument may be NULL if the returned values are to be ignored. This parameter is only used during file creation.

Return values

>=0	Newly allocated open file handle.
-EACCES	One of the following conditions occurred: • Search permission is denied on a component of the path prefix. • The file exists and the permissions specified by Oflag are denied. • The file doesn't exist and write permission is denied for the parent directory of the file to be created. • O_TRUNC is specified and write permission is denied.
-EEXIST	O_CREAT and O_EXCL are set and the named file exists.
-EFAULT	The Path parameter is a NULL pointer.
-EINPROGRESS	The file is currently being staged. The open should be retried at a later time.
-EINVAL	Oflag is not valid, or one or more values input in the HintsIn parameter is invalid.
-EISDIR	The named file is a directory. Note that opening directories via hpss_Open is not supported in any mode.
-EMFILE	The client open file table is already full.
-ENFILE	Too many files are open in the system.
-ENAMETOOLONG	The length of the Path string exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
-ENOENT	The named file does not exist and the O_CREAT flag was not specified, or the Path argument points to an empty string.
-EREMOTE	The caller requested the HPSS_O_NO_TAPE open flag, and some part of the file is only on tape.

Note

The 'hpss_Open' function establishes a connection between a file, named by 'Path', and a file handle. HPSS_O_STAGE_ASYNC and HPSS_O_STAGE_BKGRD may not be specified together.

9.3.2.7 int hpss_OpenBitfile (const bfs_bitfile_obj_handle_t ∗ *BitfileObj*, int *Oflag*, const sec_cred_t ∗ *Ucred*)

Open an HPSS file by bitfile id.

Parameters

in	*BitfileObj*	Bitfile Object Handle (usually obtained using hpss_FileGetAttributes)
in	*Oflag*	Specifies file status and file access modes to be assigned. Applicable values below may be OR'd together. • O_RDONLY • O_WRONLY • O_RDWR • O_APPEND • O_TRUNC • HPSS_O_NO_TAPE • HPSS_O_EXCL • HPSS_O_STAGE_ASYNC • HPSS_O_STAGE_BKGRD
in	*Ucred*	User Credentials

Return values

-EACCESS	The client does not have permission for the requested file access.
-EFAULT	The BitFileId parameter is a NULL pointer.
-EINPROGRESS	The file is currently being staged. The open should be retried at a later time.
-EINVAL	Oflag is not valid.
-EMFILE	The client file table is already full.
-ENFILE	Too many bitfiles are already open in the system.
-ENOENT	No entry exists for the specified bitfile id.
-EREMOTE	The caller requested the HPSS_O_NO_TAPE open flag, and some part of the file is only on tape.

Note

The 'hpss_OpenBitfile' function establishes a connection between a file, specified by the bitfile id 'BitFileID' and the bitfile hash 'BitFileHash', and a file handle.
HPSS_O_STAGE_ASYNC and HPSS_O_STAGE_BKGRD may not be specified together.

9.3.2.8 int hpss_OpenBitfileVAttrs (const hpss_vattr_t ∗ *FileAttrs*, int *Oflag*, const sec_cred_t ∗ *Ucred*, hpss_cos_hints_t ∗ *HintsOut*, uint64_t ∗ *SegmentSize*)

Open a bitfile using vattrs.

The 'hpss_OpenBitfileVAttrs' function is a no-frills interface for openning a bitfile, using an existing bitfile attributes and authorization ticket. This function is different from other open calls in that it only obtains an open context, without any extra path traverse or get attribute calls.

Parameters

`in`	*FileAttrs*	File Attributes
`in`	*Oflag*	Specifies file status and file access modes to be assigned. Applicable values below may be OR'd together. • O_RDONLY • O_WRONLY • O_RDWR • O_APPEND • O_TRUNC • HPSS_O_NO_TAPE • HPSS_O_EXCL • HPSS_O_STAGE_ASYNC • HPSS_O_STAGE_BKGRD
`in`	*Ucred*	User Credentials
`out`	*HintsOut*	Actual Hint Values Used
`out`	*SegmentSize*	Current Storage Segment Size

Returns

Upon successful completion, a nonnegative file descriptor is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

Return values

-EACCESS	The client does not have permission for the requested file access.
-EFAULT	The BitfileID parameter is a NULL pointer.
-EINPROGRESS	The file is currently being staged. The open should be retried at a later time.
-EINVAL	Oflag is not valid.
-EMFILE	The client file table is already full.
-ENFILE	Too many bitfiles are already open in the system.
-ENOENT	No entry exists for the specified bitfile id.
-EREMOTE	The caller requested the HPSS_O_NO_TAPE open flag, and some part of the file is only on tape.

Note

This function does not currently provide a partition hash to the Core Server.
HPSS_O_STAGE_ASYNC and HPSS_O_STAGE_BKGRD may not be specified together.

9.3.2.9 int hpss_OpenHandle (const **ns_ObjHandle_t** ∗ *ObjHandle,* const char ∗ *Path,* int *Oflag,* mode_t *Mode,* sec_cred_t ∗ *Ucred,* const **hpss_cos_hints_t** ∗ *HintsIn,* const **hpss_cos_priorities_t** ∗ *HintsPri,* **hpss_cos_hints_t** ∗ *HintsOut,* hpss_vattr_t ∗ *AttrsOut*)

Optionally create and open an HPSS file relative to a given directory.

Parameters

`in`	*ObjHandle*	Parent object handle; the parent directory.
`in`	*Path*	Names the file to be opened or created.
`in`	*Oflag*	Specifies the file status and access modes to be assigned. Applicable values give below below may be OR'd together. Refer to POSIX.1 for specific behavior. In addition the following HPSS flags are allowed: • HPSS_O_NO_TAPE • HPSS_O_EXCL • HPSS_O_STAGE_ASYNC • HPSS_O_STAGE_BKGRD
`in`	*Ucred*	User Credentials
`in`	*Mode*	Specifies the file mode for a file that is created as a result of O_CREAT.
`in`	*Ucred*	User credentials
`in`	*HintsIn*	Provides allocation hints to HPSS as to the expected structure or access of the file. This argument may be a NULL pointer if the default COS is to be used. This parameter is only used during file creation.
`in`	*HintsPri*	Provides the relative priorities associated with the fields contained in the HintsPri structure. This parameter is only used during file creation, and may be a NULL pointer. The priority will be set to REQUIRED_PRIORITY if HintsPri is NULL.
`in`	*HintsOut*	The hints which were used for this file open; note that these might differ from the hints provided.
`out`	*AttrsOut*	Returned file attributes.

Returns

Upon successful completion, hpss_OpenHandle returns a nonnegative value that is the newly allocated file handle. Otherwise, hpss_OpenHandle returns a negative value; the absolute value of which is equal to an errno value set by POSIX.1 open.

Return values

-EACCESS	One of the following occurred: • Search permission is denied on a component of the path prefix. • The file exists and the permissions specified by Oflag are denied. • The file does not exist and write permission is denied for the parent directory of the file to be created. • O_TRUNC is specified and write permission is denied.

-EEXIST	O_CREAT and O_EXCL are set and the named file exists.
-EFAULT	The Path parameter is a NULL pointer.
-EINPROGRESS	The file is currently being staged. The open should be retried at a later time.
-EINVAL	Oflag is not valid, or one or more values input in HintsIn are invalid.
-EISDIR	The named file is a directory. Note that opening directories via this function is not supported in any mode.
-EMFILE	The client open file table is already full.
-ENFILE	Too many open files on the system.
-ENAMETOOLONG	The length of the Path string, or a path name component exceeds the system-imposed limit.
-ENOENT	The named file does not exist and the O_CREAT flag was not specified, or the Path argument points to an empty string.
-ENOSPC	Resources could not be allocated for the new file.
-ENOTDIR	A component of the path is not a directory.
-EREMOTE	The caller requested the HPSS_O_NO_TAPE open flag, and some part of the file is only on tape.

Note

The 'hpss_OpenHandle' function establishes a connection between a file, specified by 'Path', taken relative to the directory indicated by 'ObjHandle', and a file handle.
HPSS_O_STAGE_ASYNC and HPSS_O_STAGE_BKGRD may not be specified together.

9.3.2.10 int hpss_ReopenBitfile (int *Fildes,* const **bfs_bitfile_obj_handle_t** ∗ *BitfileObj,* int *Oflag,* const sec_cred_t ∗ *Ucred*)

Open an HPSS file by bitfile object handle.

Parameters

`in`	*Fildes*	Descriptor to be reopened
`in`	*BitfileObj*	Bitfile Object to Open
`in`	*Oflag*	Specifies file status and file access modes to be assigned. Applicable values below may be OR'd together. • O_RDONLY • O_WRONLY • O_RDWR • O_APPEND • O_TRUNC • HPSS_O_NO_TAPE • HPSS_O_EXCL • HPSS_O_STAGE_ASYNC • HPSS_O_STAGE_BKGRD

in	*Ucred*	User Credentials

Returns

Upon successful completion, a nonnegative file descriptor is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

Return values

-EBADF	Filedes does not refer to an open file.
-EBUSY	The open file descriptor is in use by another thread.
-EFAULT	The BitFileId parameter is a NULL pointer.
-EINPROGRESS	The file is currently being staged. The open should be retried at a later time.
-EINVAL	Oflag does not contain a valid access mode or BitfileID is NULL.
-ENOENT	No entry exists for the specified bitfile id.
-EREMOTE	The caller requested the HPSS_O_NO_TAPE open flag, and some part of the file is only on tape.

Note

The file descriptor will not be closed on return from this routine, even if an open error occurs.
The 'hpss_ReopenBitfile' function closes an open bitfile and establishes a connection between a file, specified by the bitfile object 'BitfileObj', and the same file handle.
HPSS_O_STAGE_ASYNC and HPSS_O_STAGE_BKGRD may not be specified together.

9.3.2.11 int hpss_Unlink (const char ∗ *Path*)

Remove a namespace entry.

The 'hpss_Unlink' function removes the entry name by 'Path', and decrements the link count for the file. If link count becomes zero, the file will be deleted when it is no longer open to any client.

Parameters

in	*Path*	Path of file to unlink

Return values

0	Unlink was successful.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The file named by Path is a directory.

Note

If trashcans are enabled, Path will overwritten by the rename, and be moved to the trashcan corresponding to to the deleting user and the Path's location.

9.3.2.12 int hpss_UnlinkHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred*)

Remove a namespace entry using a handle.

The 'hpss_UnlinkHandle' function removes the entry name by 'Path', taken relative to the directory indicated by 'ObjHandle', and decrements the link count for the file. If link count becomes zero, the file will be deleted when it is no longer open to any client.

Parameters

in	*ObjHandle*	parent object handle
in	*Path*	path of file to get statistics
in	*Ucred*	user credentials

Return values

0	Unlink was successful
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	ObjHandle is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The file named by Path is a directory.

Note

If trashcans are enabled, Path will overwritten by the rename, and be moved to the trashcan corresponding to to the deleting user and the Path's location.

9.3.2.13 int hpss_UnlinkHandleImmediate (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred*)

Remove a namespace entry using a handle, bypassing the trashcan.

The 'hpss_UnlinkHandleImmediate' function removes the entry name by 'Path', taken relative to the directory indicated by 'ObjHandle', and decrements the link count for the file. If link count becomes zero, the file will be deleted when it is no longer open to any client.

Parameters

in	*ObjHandle*	parent object handle
in	*Path*	path of file to get statistics
in	*Ucred*	user credentials

Return values

0	Unlink was successful
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	ObjHandle is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The file named by Path is a directory.

Note

This API causes the trashcan to be bypassed. This API should only be used in specific circumstances where the user indicates they want to bypass being able to recover their files.

9.3.2.14 int hpss_UnlinkImmediate (const char * *Path*)

Remove a namespace entry, bypassing the trashcan.

The 'hpss_UnlinkImmediate' function removes the entry name by 'Path', and decrements the link count for the file. If link count becomes zero, the file will be deleted when it is no longer open to any client.

Parameters

`in`	*Path*	Path of file to unlink

Return values

0	Unlink was successful.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The file named by Path is a directory.

Note

This API causes the trashcan to be bypassed. This API should only be used in specific circumstances where the user indicates they want to bypass being able to recover their files.

9.4 File Data

Functions which can be used to access and update HPSS file data.

Functions

- int hpss_CopyFile (int SrcFildes, int DestFildes)

  Copies the source file to the destination file.
- int hpss_Fclear (int Fildes, uint64_t Length)

  Create a hole in an open file at its current offset.
- int hpss_FclearOffset (int Fildes, uint64_t Offset, uint64_t Length)

  Create a hole in an open file at a specified offset.
- int hpss_Fpreallocate (int Fildes, uint64_t Length)

  Preallocate storage resources for an open file.
- int hpss_Ftruncate (int Fildes, uint64_t Length)

  Truncate an open file.
- int64_t hpss_Lseek (int Fildes, int64_t Offset, int Whence)

  Sets the file offset for the open file handle.
- signed32 hpss_PreferredReadOrder (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, hpss_read_queue_list_t ∗ItemsToCheck, int Flags, int Delay, hpss_read_queue_list_t ∗ItemsReady, hpss_-read_queue_list_t ∗ItemsError)

  Request to be notified about ready items in a list of reads.
- ssize_t hpss_Read (int Fildes, void ∗Buf, size_t Nbyte)

  Reads a number of bytes from a file.
- ssize_t hpss_ReadItem (int Fildes, void ∗Buf, size_t Nbyte, hpss_read_queue_item_t ∗Item)

  Reads a number of bytes from a file using a read queue item.
- signed32 hpss_ReadQueueAdd (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, hpss-_read_queue_item_t ∗Item)

  Add an item from the provided read queue.
- signed32 hpss_ReadQueueAddMany (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, hpss_read_queue_list_t ∗ItemList)

  Add items from the provided read queue.
- signed32 hpss_ReadQueueCreateContext (uint32_t SubsysId, sec_cred_t ∗UserCredsP, hpss_uuid_t ∗ContextP)

  Create a new read queue context.
- signed32 hpss_ReadQueueList (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, hpssoid_t ∗VVID, unsigned32 Start, unsigned32 Max, hpss_read_queue_list_t ∗List)

  List items in the provided read queue.
- signed32 hpss_ReadQueueRemove (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, hpss_reqid_list_t ∗RequestList)

  Remove items from the provided read queue.
- signed32 hpss_ReadQueueRemoveContext (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗User-CredsP, int Flags)

  Remove a read queue context.
- int hpss_SetFileOffset (int Fildes, uint64_t OffsetIn, int Whence, int Direction, uint64_t ∗OffsetOut)

  Sets the file offset for an open file handle.
- int32_t hpss_SetNextIORequestID (hpss_reqid_t ∗RequestId)

  Set the request ID to be used for the next I/O in this thread. This should be a request ID generated from hpss_Get-UniqueRequestID or from hpss_ReadQueueAdd.
- int hpss_Truncate (const char ∗Path, uint64_t Length)

  Truncate a file.

- int hpss_TruncateHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, uint64_t Length, const sec-_cred_t ∗Ucred)

    Truncate a file using a handle.
- ssize_t hpss_Write (int Fildes, const void ∗Buf, size_t Nbyte)

    Write data to an open file.

9.4.1 Detailed Description

Functions which can be used to access and update HPSS file data.

9.4.2 Function Documentation

9.4.2.1 int hpss_CopyFile (int *SrcFildes*, int *DestFildes*)

Copies the source file to the destination file.

The 'hpss_CopyFile' function copies data from an open source file, *SrcFildes* to an open destination file, *DestFildes*. This does not include any metadata information, time information, etc.

Parameters

in	*SrcFildes*	open desc for src file
in	*DestFildes*	open desc for dest file

Return values

0	File successfully copied
-EBADF	Source or destination file descriptors out of range (i.e. negative) or not HPSS type file descriptors, or the destination file was not opened for writes.
-EBUSY	Another thread is currently manipulating the destination file.
-EIO	An internal HPSS error has occurred.
-ENOTSUP	Source and destination files not owned by same Core Server.
-ESTALE	Connection for destination file no longer valid.

9.4.2.2 int hpss_Fclear (int *Fildes*, uint64_t *Length*)

Create a hole in an open file at its current offset.

The 'hpss_Fclear' function creates a hole in a file, beginning at its current file position, for the specified length. The storage resources associated with this hole may be freed accordingly.

Parameters

in	*Fildes*	Specifies the file descriptor identifying the open file
in	*Length*	Number of bytes to be cleared

Return values

0	Success
-EBADF	The specified file descriptor does not correspond to a file opened for writing.

-EBUSY	The specified file descriptor is currently in use.

See Also

hpss_FclearOffset hpss_Ftruncate

9.4.2.3 int hpss_FclearOffset (int *Fildes*, uint64_t *Offset*, uint64_t *Length*)

Create a hole in an open file at a specified offset.

The 'hpss_FclearOffset' function sets creates a hole in a file beginning at the specified Offset for the specified length. Storage resources associated with the area where the hole was made may be freed accordingly.

Parameters

`in`	*Fildes*	Specifies the file descriptor identifying the open file
`in`	*Offset*	Specifies where to begin clearing the file
`in`	*Length*	Specifies the number of bytes to be cleared

Return values

0	Success
-EINVAL	The *Offset* or *Length* is valid
-EBADF	The specified file descriptor does not correspond to a file opened for writing.
-EBUSY	The specified file descriptor is currently in use.

See Also

hpss_Fclear

9.4.2.4 int hpss_Fpreallocate (int *Fildes*, uint64_t *Length*)

Preallocate storage resources for an open file.

The 'hpss_Fpreallocate' function sets the file length for the open file associated with the handle, *Fildes* and preallocates storage. It must be greater than the current size of the file.

Parameters

`in`	*Fildes*	ID of open object
`in`	*Length*	New length

Return values

0	Success
-EBADF	The specified file descriptor does not correspond to a file opened for writing.
-EBUSY	The specified file descriptor is currently in use.
-ENOSPC	The requested storage resources could not be allocated.
-EINVAL	There is not a disk storage class at the top of the storage hierarchy.

Note

There must be a disk storage class at the top of the storage hierarchy in which the file resides.

9.4.2.5 int hpss_Ftruncate (int *Fildes,* uint64_t *Length*)

Truncate an open file.

The 'hpss_Ftruncate' function sets the file length for the open file associated with the handle, *Fildes*. If the new file length is greater than the current length, a hole is created in the file.

Parameters

`in`	*Fildes*	ID of open object
`in`	*Length*	New length

Return values

0	Success
-EBADF	The specified file descriptor does not correspond to a file opened for writing.
-EBUSY	The specified file descriptor is currently in use.
-EINVAL	No DMAP support compiled in.

Warning

Data will be lost if the file is truncated to a size less than the last written byte. This function should be used with care.

9.4.2.6 int64_t hpss_Lseek (int *Fildes,* int64_t *Offset,* int *Whence*)

Sets the file offset for the open file handle.

The 'hpss_Lseek' function sets the file offset for the open file handle, 'Fildes'.

Parameters

`in`	*Fildes*	ID of open object
`in`	*Offset*	# of bytes to calculate new offset
`in`	*Whence*	Origin for the seek

Returns

Upon successful completion hpss_Lseek returns a non-negative value representing the resulting offset as measured in bytes from the beginning of the file.

Return values

-EBADF	The specified file descriptor does not refer to an open file.
-EBUSY	The file is currently in use by another client thread.
-EFBIG	Could not represent the resulting offset in the return value.
-EINVAL	The Whence parameter is invalid or the resulting offset would be invalid.
-ESTALE	Connection to this file descriptor is no longer valid.

9.4.2.7 signed32 hpss_PreferredReadOrder (uint32_t *SubsysId,* hpss_uuid_t ∗ *ContextP,* sec_cred_t ∗ *UserCredsP,* hpss_read_queue_list_t ∗ *ItemsToCheck,* int *Flags,* int *Delay,* hpss_read_queue_list_t ∗ *ItemsReady,* hpss_read_queue_list_t ∗ *ItemsError*)

Request to be notified about ready items in a list of reads.

Parameters

in	*SubsysId*	Subsystem ID
in	*ContextP*	Read context to use
in	*UserCredsP*	User Credentials
in	*ItemsToCheck*	List of read requests to check on (or NULL)
in	*Flags*	Flags • CORE_RQ_ONLY_NEW_READY. Only return new, ready read requests.
in	*Delay*	How long to check for requests which are ready to run
out	*ItemsReady*	Which items are ready to run
out	*ItemsError*	Items which have errored out

This function will block up to Delay seconds. The ItemsToCheck should all be against the same VV. The items will be checked periodically up to Delay seconds. At the point where any items in ItemsToCheck are ready to run, the function will include those in ItemsReady in the order they will be run in. If ItemsToCheck is NULL, then this function will return any items which are ready.

Return values

0	Success
-EFAULT	One or more of ContextP, ItemsReady, or ItemsError is NULL
-ENOENT	ItemsToCheck has no entries, or nothing is ready
Other	POSIX error indicating the nature of the error

Note

The CORE_RQ_ONLY_NEW_READY flag causes the core server to only return read queue items which are ready but have not previously been returned as ready across all clients.

9.4.2.8 ssize_t hpss_Read (int *Fildes,* void ∗ *Buf,* size_t *Nbyte*)

Reads a number of bytes from a file.

Parameters

in	*Fildes*	ID of object to be read
in	*Buf*	Buffer in which to receive data
in	*Nbyte*	number of bytes to read

The 'hpss_Read' function attempts to read 'Nbyte' bytes from a file for open handle, 'Fildes', into the client buffer pointed to by 'Buf'.

Return values

positive	Number of Bytes Read
0	End of file reached
-EBADF	The specified file descriptor does not correspond to a file opened for reading.
-EBUSY	The file is currently in use by another client thread.
-EFAULT	The Buf parameter is NULL.
-EIO	An input/output or HPSS internal error occurred.

9.4.2.9 ssize_t hpss_ReadItem (int *Fildes,* void ∗ *Buf,* size_t *Nbyte,* hpss_read_queue_item_t ∗ *Item*)

Reads a number of bytes from a file using a read queue item.

This function behaves the exact same as hpss_Read, except that it connects the read request with the specified read queue item. See hpss_ReadQueueAdd for more information.

Parameters

`in`	*Fildes*	ID of object to be read
`in`	*Buf*	Buffer in which to receive data
`in`	*Nbyte*	number of bytes to read
`in`	*Item*	Read queue item

The 'hpss_ReadItem' function attempts to read 'Nbyte' bytes from a file for open handle, 'Fildes', into the client buffer pointed to by 'Buf'.

Return values

positive	Number of Bytes Read
0	End of file reached
-EBADF	The specified file descriptor does not correspond to a file opened for reading.
-EBUSY	The file is currently in use by another client thread.
-EFAULT	The Buf or Item parameter is NULL.
-EIO	An input/output or HPSS internal error occurred.
-EIO	Failed to set the request ID
Other	POSIX error indicating the nature of the error

9.4.2.10 signed32 hpss_ReadQueueAdd (uint32_t *SubsysId*, hpss_uuid_t ∗ *ContextP*, sec_cred_t ∗ *UserCredsP*, hpss_read_queue_item_t ∗ *Item*)

Add an item from the provided read queue.

Parameters

`in`	*SubsysId*	Subsystem ID
`in`	*ContextP*	Read context to add to
`in`	*Item*	Read request to add

Return values

0	Success
-EFAULT	ContextP or Item is NULL
-ENOMEM	Out of memory
Other	POSIX error indicating the nature of the error

9.4.2.11 signed32 hpss_ReadQueueAddMany (uint32_t *SubsysId*, hpss_uuid_t ∗ *ContextP*, sec_cred_t ∗ *UserCredsP*, hpss_read_queue_list_t ∗ *ItemList*)

Add items from the provided read queue.

Parameters

`in`	*SubsysId*	Subsystem ID
`in`	*ContextP*	Read context to add to
`in`	*ItemList*	Read request to add

Return values

0	Success

-EFAULT	ContextP or ItemList are NULL
-EINVAL	ItemList is empty
Other	POSIX error indicating the nature of the error

9.4.2.12 **signed32** hpss_ReadQueueCreateContext (uint32_t *SubsysId,* sec_cred_t ∗ *UserCredsP,* hpss_uuid_t ∗ *ContextP*)

Create a new read queue context.

Parameters

`in`	*SubsysId*	Subsystem ID
`out`	*ContextP*	Context UUID

Return values

0	Success
-EFAULT	ContextP is NULL
Other	POSIX error indicating the nature of the error

9.4.2.13 **signed32** hpss_ReadQueueList (uint32_t *SubsysId,* hpss_uuid_t ∗ *ContextP,* sec_cred_t ∗ *UserCredsP,* hpssoid_t ∗ *VVID,* **unsigned32** *Start,* **unsigned32** *Max,* **hpss_read_queue_list_t** ∗ *List*)

List items in the provided read queue.

Parameters

`in`	*SubsysId*	Subsystem ID
`in`	*ContextP*	Read context to use
`in`	*UserCredsP*	User Credentials
`in`	*VVID*	VV to List (optional)
`in`	*Start*	Initial list index to start listing from in ContextP
`in`	*Max*	Maximum number of items to return
`out`	*List*	Sorted list of entries in the VVIDs read queue

Return values

0	Success
-EFAULT	ContextP or List is NULL
Other	POSIX error indicating the nature of the error

9.4.2.14 **signed32** hpss_ReadQueueRemove (uint32_t *SubsysId,* hpss_uuid_t ∗ *ContextP,* sec_cred_t ∗ *UserCredsP,* **hpss_reqid_list_t** ∗ *RequestList*)

Remove items from the provided read queue.

Parameters

`in`	*SubsysId*	Subsystem ID
`in`	*ContextP*	Read context to use
`in`	*UserCredsP*	User Credentials

in	*RequestList*	List of requests to remove from the queue

Return values

0	Success
-EFAULT	Context or RequestList is NULL
Other	POSIX error indicating the nature of the error

9.4.2.15 signed32 hpss_ReadQueueRemoveContext (uint32_t *SubsysId,* hpss_uuid_t ∗ *ContextP,* sec_cred_t ∗ *UserCredsP,* int *Flags*)

Remove a read queue context.

Parameters

in	*SubsysId*	Subsystem ID
in	*ContextP*	Read queue context to use
in	*Flags*	Read Context Flags • HPSS_REMOVE_CONTEXT_ABORT - Abort all existing requests

Return values

0	Success
-EFAULT	ContextP is NULL
Other	POSIX error indicating the nature of the error

9.4.2.16 int hpss_SetFileOffset (int *Fildes,* uint64_t *OffsetIn,* int *Whence,* int *Direction,* uint64_t ∗ *OffsetOut*)

Sets the file offset for an open file handle.

The ’hpss_SetFileOffset’ function sets the file offset for the open file handle, ’Fildes’. This function allows a 64-bit offset value to be specified and returned (via OffsetOut). The ’Direction’ argument is provided to allow for negative offfsets (ala lseek()).

Parameters

in	*Fildes*	ID of open object
in	*OffsetIn*	# of bytes to calculate new offset
in	*Whence*	Origin for the seek
in	*Direction*	OffsetIn is either forward or backward
out	*OffsetOut*	Resulting offset

Returns

Upon successful completion hpss_SetFileOffset returns zero and a value representing the resulting offset as measured in bytes from the beginning of the file.

Return values

-EBADF	The specified file descriptor does not refer to an open file.

-EBUSY	The file is currently in use by another client thread.
-EFAULT	OffsetOut is a NULL pointer.
-EINVAL	The Whence or Direction parameter is invalid or the resulting offset would be invalid such as a negative value or beyond the largest supported file size.
-ENOSPC	Resources could not be allocated to satisfy the request.

9.4.2.17 int32_t hpss_SetNextIORequestID (hpss_reqid_t * *RequestId*)

Set the request ID to be used for the next I/O in this thread. This should be a request ID generated from hpss_Get-UniqueRequestID or from hpss_ReadQueueAdd.

Sets the next request id

Return values

HPSS_E_NOERROR	Success
EFAULT	NULL Request ID provided
EINVAL	Invalid request ID provided

Note

This should not be called except with a request ID provided via the hpss_ReadQueueAddItem or hpss_Read-QueueAddMany APIs.

9.4.2.18 int hpss_Truncate (const char * *Path,* uint64_t *Length*)

Truncate a file.

The 'hpss_Truncate' function sets the file length for the file name by 'path'.

Parameters

`in`	*Path*	Pathname of file
`in`	*Length*	New length

Return values

0	Success
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the file.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	Path specifies a directory.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The specified path name does not exist.
-ENOTDIR	A component of the Path prefix is not a directory.

Warning

Data will be lost if the file is truncated to a size less than the last written byte. This function should be used with care.

9.4.2.19 int hpss_TruncateHandle (const ns_ObjHandle_t * *ObjHandle*, const char * *Path*, uint64_t *Length*, const sec_cred_t * *Ucred*)

Truncate a file using a handle.

The 'hpss_TruncateHandle' function sets the file length for the file specified by 'Handle'.

Parameters

in	*ObjHandle*	Handle to file
in	*Path*	file name
in	*Length*	New length
in	*Ucred*	user credentials

Return values

0	Success
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the file.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	Path specifies a directory or ObjHandle is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The specified path name does not exist.
-ENOTDIR	A component of the Path prefix is not a directory.

Warning

Data will be lost if the file is truncated to a size less than the last written byte. This function should be used with care.

9.4.2.20 ssize_t hpss_Write (int *Fildes*, const void * *Buf*, size_t *Nbyte*)

Write data to an open file.

The 'hpss_Write' function attempts to write 'Nbyte' bytes of data from the client buffer pointed to by 'Buf' to the file associated with the open file handle, 'Fildes'.

Parameters

in	*Fildes*	ID of object to be written
in	*Buf*	Buffer from which to send data
in	*Nbyte*	number of bytes to write

Return values

0	End of File reached (EOF)
>0	Number of bytes successfully written
-EBADF	The specified file descriptor does not correspond to a file opened for writing.
-EBUSY	The file is currently in use by another client thread.
-EFAULT	The Buf parameter is out of range.
-EFBIG	The write operation would cause the file to exceed the system-imposed maximum file length.

-EIO	An input/output or HPSS internal error occurred.
-ENOSPC	There is no free space remaining to satisfy the write request.

9.5 Fileset and Junction

Functions which operate upon filesets and junctions.

Functions

- int hpss_FilesetCreate (const hpss_srvr_id_t ∗CoreServerID, uint32_t CreateOptions, ns_FilesetAttrBits-_t FilesetAttrBits, const ns_FilesetAttrs_t ∗FilesetAttrs, hpss_AttrBits_t ObjectAttrBits, const hpss_Attrs_-t ∗ObjectAttrs, ns_FilesetAttrBits_t RetFilesetAttrBits, hpss_AttrBits_t RetObjectAttrBits, ns_FilesetAttrs_t ∗RetFilesetAttrs, hpss_Attrs_t ∗RetObjectAttrs, ns_ObjHandle_t ∗FilesetHandle)

    Creates a new fileset.
- int hpss_FilesetDelete (const char ∗FilesetName, const uint64_t ∗FilesetId, const ns_ObjHandle_t ∗Fileset-Handle)

    Deletes an existing fileset.
- int hpss_FilesetGetAttributes (const char ∗FilesetName, const uint64_t ∗FilesetId, const ns_ObjHandle_-t ∗FilesetHandle, const hpss_srvr_id_t ∗CoreServerID, ns_FilesetAttrBits_t FilesetAttrBits, ns_FilesetAttrs_t ∗FilesetAttrs)

    Retrieves the attributes of a fileset.
- int hpss_FilesetListAll (uint64_t OffsetIn, uint32_t Entries, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_global_-fsent_t ∗FSentPtr)

    Retrieves attributes for every HPSS fileset.
- int hpss_FilesetSetAttributes (const char ∗FilesetName, const uint64_t ∗FilesetId, const ns_ObjHandle_t ∗FilesetHandle, ns_FilesetAttrBits_t FilesetAttrBitsIn, const ns_FilesetAttrs_t ∗FilesetAttrsIn, ns_FilesetAttr-Bits_t FilesetAttrBitsOut, ns_FilesetAttrs_t ∗FilesetAttrsOut)

    Sets the attributes of a fileset.
- int hpss_GetJunctionAttrs (const char ∗Path, hpss_Attrs_t ∗AttrOut)

    Retrieve file statistics associated with directory listing operations including junctions.
- int hpss_GetJunctions (uint32_t SubsystemID, uint64_t OffsetIn, uint32_t Entries, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_junction_ent_t ∗JentPtr)

    Retrives the junctions that are managed by the Core Server.
- int hpss_JunctionCreate (const char ∗Path, const ns_ObjHandle_t ∗SourceHandle, ns_ObjHandle_t ∗JunctionHandle)

    Creates an HPSS junction.
- int hpss_JunctionCreateHandle (const ns_ObjHandle_t ∗ParentHandle, const char ∗Path, const ns_Obj-Handle_t ∗SourceHandle, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗JunctionHandle)

    Creates an HPSS junction in relation to a fileset handle.
- int hpss_JunctionDelete (const char ∗Path)

    Deletes a junction.
- int hpss_JunctionDeleteHandle (const ns_ObjHandle_t ∗ParentHandle, const char ∗Path, const sec_cred_t ∗Ucred)

    Deletes a junction.

9.5.1 Detailed Description

Functions which operate upon filesets and junctions.

9.5.2 Function Documentation

9.5.2.1 int hpss_FilesetCreate (const **hpss_srvr_id_t** ∗ *CoreServerID,* uint32_t *CreateOptions,* **ns_FilesetAttrBits_t** *FilesetAttrBits,* const **ns_FilesetAttrs_t** ∗ *FilesetAttrs,* hpss_AttrBits_t *ObjectAttrBits,* const **hpss_Attrs_t** ∗ *ObjectAttrs,* **ns_FilesetAttrBits_t** *RetFilesetAttrBits,* hpss_AttrBits_t *RetObjectAttrBits,* **ns_FilesetAttrs_t** ∗ *RetFilesetAttrs,* **hpss_Attrs_t** ∗ *RetObjectAttrs,* **ns_ObjHandle_t** ∗ *FilesetHandle*)

Creates a new fileset.

Deprecated The *CreateOptions* parameter of this function will be removed in a future version of HPSS.

The 'hpss_FilesetCreate' function is called to create a new fileset.

Parameters

`in`	*CoreServerID*	Core Server ID
`in`	*CreateOptions*	creation options CORE_NS_FILESET - Create Core Server fileset metadata
`in`	*FilesetAttrBits*	Fileset attributes to set
`in`	*FilesetAttrs*	Fileset attributes
`in`	*ObjectAttrBits*	Object attributes to set
`in`	*ObjectAttrs*	Object attributes
`out`	*RetFilesetAttr-Bits*	Fileset attr to return
`out`	*RetObjectAttr-Bits*	Dir attr to return
`out`	*RetFilesetAttrs*	Returned fileset attr
`out`	*RetObjectAttrs*	Returned dir attr
`out`	*FilesetHandle*	New fileset handle

Return values

0	Success
-EACCES	The user is not the root user or a trusted user.
-EFAULT	The CoreServerID, FilesetHandle, FilesetAttrs, ObjectAttrs, RetFilesetAttrs or the RetObjectAttrs is NULL.
-EINVAL	The file attributes or attributes bits are invalid.
-EEXIST	A file already exist with the specified identifier.

Note

If a NULL 'NameServer' ID is specified the root Name Server will be used.
Two other fileset options exist but are no longer supported.

9.5.2.2 int hpss_FilesetDelete (const char ∗ *FilesetName,* const uint64_t ∗ *FilesetId,* const **ns_ObjHandle_t** ∗ *FilesetHandle*)

Deletes an existing fileset.

The 'hpss_FilesetDelete' function is called to delete an existing fileset by either name, id or handle.

Parameters

`in`	*FilesetName*	Fileset name
`in`	*FilesetId*	Fileset ID

in	*FilesetHandle*	Fileset object handle

Return values

0	Success
-EACCES	The user is not the root user or a trusted user.
-EFAULT	The Name, FilesetID and FilesetHandle arguments are all NULL pointers.
-EINVAL	More that one type of fileset identifier was specified.
-ENOENT	The specified fileset does not exist.

Note

A fileset can be identified by either a name, an ID, or the handle to its root. Only one type of identifier is valid. If more than one of the fileset identifiers have non-null pointers, then the call will fail with an EINVAL.

9.5.2.3 int hpss_FilesetGetAttributes (const char ∗ *FilesetName,* const uint64_t ∗ *FilesetId,* const ns_ObjHandle_t ∗ *FilesetHandle,* const hpss_srvr_id_t ∗ *CoreServerID,* ns_FilesetAttrBits_t *FilesetAttrBits,* ns_FilesetAttrs_t ∗ *FilesetAttrs*)

Retrieves the attributes of a fileset.

The 'hpss_FilesetGetAttributes' function is called to retrieve the attribute for a specified Core Server fileset by either name, id or handle. If NULL is specified for the Core Server id, then the local Root Core Server will be contacted; otherwise, the specified Root Core Server will be contacted to retrieve the fileset information.

Parameters

in	*FilesetName*	Fileset name
in	*FilesetId*	Fileset ID
in	*FilesetHandle*	Fileset object handle
in	*CoreServerID*	Core server ID for fileset
in	*FilesetAttrBits*	Fileset attribute bits
out	*FilesetAttrs*	Returned attributes

Return values

-EACCES	The user is not the root user or a trusted user.
-EFAULT	The FilesetName, FilesetID and FilesetHandle arguments are all NULL pointers.
-EINVAL	More that one type of fileset identifier was specified or invalid file set attribute bits were specified.
-ENOENT	The specified fileset does not exist.

Note

A fileset can be identified by either a name, an ID, or the handle to its root. Only one type of identifier is valid. If more than one of the fileset identifiers have non-null pointers, then the call will fail with an EINVAL.

9.5.2.4 int hpss_FilesetListAll (uint64_t *OffsetIn,* uint32_t *Entries,* uint32_t ∗ *End,* uint64_t ∗ *OffsetOut,* hpss_global_fsent_t ∗ *FSentPtr*)

Retrieves attributes for every HPSS fileset.

The 'hpss_FilesetListAll' function is called to get the global fileset attributes for all the filesets in the HPSS site.

Parameters

in	*OffsetIn*	offset to start at
in	*Entries*	number of entries to return
out	*End*	hit end of the list
out	*OffsetOut*	offset of last entry
out	*FSentPtr*	fileset entry information

Return values

0	Success
-EFAULT	The End, OffsetOut or FSentPtr parameter is a NULL pointer.
-EINVAL	The Entries parameter specifies zero entries to read.

Note

This API is used by setting 'OffsetIn' to the starting point for the lookup (usually zero). 'Entries' is the number of filesets entries for which you have allocate space. 'OffsetOut' is the point that the lookup as at when it accumulated the specified number of entires. This is typically used to specify the new starting offset ('OffsetIn'). 'End' is a flag indicating that the end of the list was encountered before the all the entries were accumulated.

9.5.2.5 int hpss_FilesetSetAttributes (const char ∗ *FilesetName,* const uint64_t ∗ *FilesetId,* const ns_ObjHandle_t ∗ *FilesetHandle,* ns_FilesetAttrBits_t *FilesetAttrBitsIn,* const ns_FilesetAttrs_t ∗ *FilesetAttrsIn,* ns_FilesetAttrBits_t *FilesetAttrBitsOut,* ns_FilesetAttrs_t ∗ *FilesetAttrsOut*)

Sets the attributes of a fileset.

The 'hpss_FilesetSetAttributes' function is called to set the attributes for a specified Core Server fileset by either name, id or handle.

Parameters

in	*FilesetName*	Fileset name
in	*FilesetId*	Fileset ID
in	*FilesetHandle*	Fileset object handle
in	*FilesetAttrBitsIn*	Fileset attribute bits
in	*FilesetAttrsIn*	Fileset attributes
in	*FilesetAttrBits-Out*	Returned set attribute bits
out	*FilesetAttrsOut*	Returned attributes

Return values

0	Success
-EACCES	The user is not the root user or a trusted user.
-EFAULT	The Name, FilesetID and FilesetHandle arguments are all NULL pointers.
-EINVAL	More that one type of fileset identifier was specified or invalid file set attributes (or attribute bits) were specified.
-ENOENT	The specified fileset does not exist.

Note

A fileset can be identified by either a name, an ID, or the handle to its root. Only one type of identifier is valid. If more than one of the fileset identifiers have non-null pointers, then the call will fail with an EINVAL.

9.5.2.6 int hpss_GetJunctionAttrs (const char * *Path,* hpss_Attrs_t * *AttrOut*)

Retrieve file statistics associated with directory listing operations including junctions.

The 'hpss_GetJunctionAttrs' function obtains information needed to support directory listing operations for the file or directory named by 'Path' and returns the information in a structure pointed to by 'AttrOut'. If 'Path' refers to a junction or symbolic link, information will be returned about the junction or link itself rather than the target object.

Parameters

in	*Path*	path of file to get statistics
out	*AttrOut*	Returned statistics

Return values

0	No error. Statistics in buffer.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path or AttrOut parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

9.5.2.7 int hpss_GetJunctions (uint32_t *SubsystemID,* uint64_t *OffsetIn,* uint32_t *Entries,* uint32_t * *End,* uint64_t * *OffsetOut,* hpss_junction_ent_t * *JentPtr*)

Retrives the junctions that are managed by the Core Server.

The 'hpss_GetJunctions' function is called to get an array of junctions that are being managed by the specified Core Server.

Parameters

in	*SubsystemID*	Subsystem identifier
in	*OffsetIn*	offset to start at
in	*Entries*	number of entries to return
out	*End*	hit end of directory
out	*OffsetOut*	offset of last entry
out	*JentPtr*	junction entry information

Return values

0	Success
-EFAULT	The End, OffsetOut or JentPtr parameter is a NULL pointer.
-EINVAL	The Entries parameter specifies zero entries to read.

9.5.2.8 int hpss_JunctionCreate (const char * *Path,* const ns_ObjHandle_t * *SourceHandle,* ns_ObjHandle_t * *JunctionHandle*)

Creates an HPSS junction.

The 'hpss_JunctionCreate' function is called to create a HPSS junction to the specified directory or fileset handle.

Parameters

in	*Path*	Path of the new junction
in	*SourceHandle*	Directory or fileset handle
out	*JunctionHandle*	Junction handle

Returns

The object handle for the new junction

Return values

0	Success
-EFAULT	Either the Path, SourceHandle, or JunctionHandle parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system imposed limit, or a component of the pathname exceeds the system imposed limit.
-ENOENT	The Path argument points to an empty string.
-EEXIST	The named path already exists in the HPSS name space.
-EACCES	The requesting client is not the root user or a trusted user with write permissions.

9.5.2.9 int hpss_JunctionCreateHandle (const ns_ObjHandle_t ∗ *ParentHandle,* const char ∗ *Path,* const ns_ObjHandle_t ∗ *SourceHandle,* const sec_cred_t ∗ *Ucred,* ns_ObjHandle_t ∗ *JunctionHandle*)

Creates an HPSS junction in relation to a fileset handle.

The 'hpss_JunctionCreateHandle' function is called to create a HPSS junction to the specified directory or fileset handle.

Parameters

in	*ParentHandle*	directory to insert junction
in	*Path*	path name of the new junction
in	*SourceHandle*	Directory or fileset handle
in	*Ucred*	user credentials
out	*JunctionHandle*	Junction handle

Returns

The object handle for the new junction

Return values

0	Success
-EFAULT	Either the Path, SourceHandle, or JunctionHandle parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system imposed limit, or a component of the pathname exceeds the system imposed limit.
-ENOENT	The Path argument points to an empty string.
-EEXIST	The named path already exists in the HPSS name space.
-EACCES	The requesting client is not the root user or a trusted user with write permissions.
-EINVAL	The SourceHandle parameter doesn't point to a directory handle or the Path is invalid.

9.5.2.10 int hpss_JunctionDelete (const char ∗ *Path*)

Deletes a junction.

The 'hpss_JunctionDelete' function is called to delete a junction.

Parameters

in	*Path*	Path of junction to delete

Return values

0	Success
-ENOENT	The Path parameter is an empty string or doesn't refer to an existing object.
-EFAULT	The Path parameter is NULL.
-EINVAL	The Path parameter doesn't refer to a junction.
-EACCES	The requesting client is not the root user or a trusted user with write permissions.

Note

If trashcans are enabled, the junction is moved to the trashcan corresponding to the deleting user and the junction's location.

9.5.2.11 int hpss_JunctionDeleteHandle (const ns_ObjHandle_t ∗ *ParentHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred*)

Deletes a junction.

The 'hpss_JunctionDeleteHandle' function is called to delete a junction specified by 'ParentHandle' and 'Path'.

Parameters

in	*ParentHandle*	Parent directory
in	*Path*	Name of junction
in	*Ucred*	user credentials

Return values

0	Success
-ENOENT	The Path parameter is an empty string or doesn't refer to an existing object
-EFAULT	The Path parameter is NULL.
-EINVAL	The Path parameter doesn't refer to a junction or the ParentHandle is NULL.
-EACCES	The requesting client is not the root user or a trusted user with write permissions.

Note

If trashcans are enabled, the junction is moved to the trashcan corresponding to the deleting user and the junction's location.

9.6 File Attribute

Functions which operate upon HPSS file attributes.

Functions

- int hpss_Access (const char ∗Path, int Amode)

  Checks the accessibility of a file.
- int hpss_AccessHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, int Amode, const sec_cred_t ∗Ucred)

  Checks the accessibility of a file using a handle.
- int hpss_Chacct (const char ∗Path, acct_rec_t AcctCode)

  Changes a file or directory's account code.
- int hpss_ChacctByName (const char ∗Path, const char ∗AcctName)

  Changes a file or directory's account name.
- int hpss_Chmod (const char ∗Path, mode_t Mode)

  Alters a file or directory's permissions.
- int hpss_Chown (const char ∗Path, uid_t Owner, gid_t Group)

  Changes the user id and group id of a file or directory.
- int hpss_FileGetAttributes (const char ∗Path, hpss_fileattr_t ∗AttrOut)

  Queries the attributes of a file.
- int hpss_FileGetAttributesBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, char ∗Path, hpss_fileattr_t ∗AttrOut)

  Get file attributes using a Bitfile Object.
- int hpss_FileGetAttributesHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, hpss_fileattr_t ∗AttrOut)

  Queries the attributes of a file using a handle.
- int hpss_FileGetXAttributes (const char ∗Path, uint32_t Flags, uint32_t StorageLevel, hpss_xfileattr_t ∗AttrOut)

  Queries the attributes of a file using flags and storage level perms.
- int hpss_FileGetXAttributesHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t Flags, uint32_t StorageLevel, hpss_xfileattr_t ∗AttrOut)

  Queries the attributes of a file using a handle, flags, and a storage level.
- int hpss_FileSetAttributes (const char ∗Path, hpss_fileattrbits_t SelFlags, const hpss_fileattr_t ∗AttrIn, hpss_fileattr_t ∗AttrOut)

  Changes the attributes of an object.
- int hpss_FileSetAttributesBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, hpss_fileattrbits_t SelFlags, const hpss_fileattr_t ∗AttrIn, hpss_fileattr_t ∗AttrOut, char ∗Path)

  Change the attributes of an object specified by bitfile object.
- int hpss_FileSetAttributesHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, hpss_fileattrbits_t SelFlags, const hpss_fileattr_t ∗AttrIn, hpss_fileattr_t ∗AttrOut)

  Changes the attributes of an object based upon an object handle and path.
- int hpss_FileSetCOS (const char ∗Path, uint32_t COSId, uint32_t StreamId)

  Change the COS of an file.
- int hpss_FileSetCOSHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t COSId, uint32_t StreamId)

  Change the COS of a file.
- int hpss_Fstat (int Fildes, hpss_stat_t ∗Buf)

  Retrieve statistics for an open file.
- int hpss_GetAttrHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗HandleOut, hpss_vattr_t ∗AttrOut)

Obtains information about a file or directory using a handle.

- int hpss_GetFileNotrunc (const char ∗Path, int ∗NotruncFlag)

  Retrieves the file's NOTRUNC_FINAL_SEG flag.
- int hpss_GetListAttrs (const char ∗Path, hpss_Attrs_t ∗AttrOut)

  Retrieve file statistics associated with directory listing operations.
- int hpss_GetRawAttrHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗HandleOut, hpss_vattr_t ∗AttrOut)

  Obtains information about a symlink or junction using a handle.
- int hpss_Lstat (const char ∗Path, hpss_stat_t ∗Buf)

  Retrieve file, or link, statistics.
- int hpss_PurgeLock (int Fildes, purgelock_flag_t Flag)

  Locks or unlocks a file's purge state.
- int hpss_PurgeOnMigrate (int Fildes, purgeonmigrate_flag_t Flag)

  Sets or clears the purge on migrate flag.
- int hpss_SetAttrHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint64_t SelFlagsIn, const hpss_vattr_t ∗AttrIn, uint64_t ∗SelFlagsOut, hpss_vattr_t ∗AttrOut)

  Updates attribute values for a file or directory.
- int hpss_SetCOSByHints (int Fildes, uint32_t Flags, const hpss_cos_hints_t ∗HintsPtr, const hpss_cos_priorities_t ∗PrioPtr, hpss_cos_md_t ∗COSPtr)

  Sets the COS of an empty file.
- int hpss_SetFileNotrunc (const char ∗Path, notrunc_flag_t Flag)

  Sets or clears a file's NOTRUNC_FINAL_SEG flag.
- int hpss_Stat (const char ∗Path, hpss_stat_t ∗Buf)

  Retrieve file statistics.
- int hpss_Utime (const char ∗Path, const struct utimbuf ∗Times)

  Set the access and modification times of a file.

9.6.1 Detailed Description

Functions which operate upon HPSS file attributes.

9.6.2 Function Documentation

9.6.2.1 int hpss_Access (const char ∗ *Path,* int *Amode*)

Checks the accessibility of a file.

The hpss_Access function checks the accessibility of the file named by *Path* for the file access indicated by *Amode*. Refer to POSIX.1 for more detailed information.

Parameters

`in`	*Path*	Path of file to check access rights
`in`	*Amode*	Indicates the type of file access being checked. Refer to POSIX.1 for possible values.

Return values

0	No error, caller has access.

-EACCES	The permissions specified by Amode are denied, or search permission is denied on a component path of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	An invalid value was specified for Amode.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

See Also

hpss_AccessHandle, hpss_Chmod, hpss_FileSetAttributes

9.6.2.2 int hpss_AccessHandle (const **ns_ObjHandle_t** ∗ *ObjHandle,* const char ∗ *Path,* int *Amode,* const sec_cred_t ∗ *Ucred*)

Checks the accessibility of a file using a handle.

The hpss_AccessHanlde function checks the accessibility of the file *Path* for the file access indicated by *Amode*. Refer to POSIX.1 for more detailed information.

Parameters

in	*ObjHandle*	Handle to the parent object
in	*Path*	Path name of the file being checked
in	*Amode*	Indicates the type of file access being checked. Refer to POSIX.1 for possible values.
in	*Ucred*	User credentials

Return values

0	No error, caller has access.
-EACCES	The permissions specified by *Amode* are denied, or search permission is denied on a component of the path prefix.
-EINVAL	An invalid vaue was specified for *Amode* or the Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the *Path* argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the *Path* argument points to an empty string.
-ENOTDIR	A component of the *Path* prefix is not a directory.

See Also

hpss_Access, hpss_Chmod, hpss_FileSetAttributes

9.6.2.3 int hpss_Chacct (const char ∗ *Path,* **acct_rec_t** *AcctCode*)

Changes a file or directory's account code.

The 'hpss_Chacct' function calls alters the account code associated with the file or directory named by *Path*.

Parameters

in	*Path*	Name of the file whose account code will be modified
in	*AcctCode*	New accounting code for the file at *Path*

Return values

-EACCES	Search permission is denied on a component of the *Path* prefix.
-EFAULT	*Path* is a NULL pointer.
-ENAMETOOLONG	The length of the *Path* argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the *Path* argument points to an empty string.
-ENOTDIR	A component of the *Path* prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation or is configured for Unix-style accounting.
0	No error. Account code altered as directed.

See Also

hpss_ChacctByName

9.6.2.4 int hpss_ChacctByName (const char ∗ *Path*, const char ∗ *AcctName*)

Changes a file or directory's account name.

The 'hpss_ChacctByName' function alters the account name associated with the file or directory named by *Path* using the account name specified by *AcctName*.

Parameters

in	*Path*	File or directory for which the account code is being changed
in	*AcctName*	Account name corresponding to the new account code for the file or directory

Return values

0	No error. Account name altered as directed.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	Path is a NULL pointer or AcctName is NULL.
-EINVAL	The specified AccountName is not a valid account name.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation or is configured for Unix-style accounting.

9.6.2.5 int hpss_Chmod (const char ∗ *Path*, mode_t *Mode*)

Alters a file or directory's permissions.

The 'hpss_Chmod' function alters the file or directory mode associated with the file or directory named by *Path*.

Parameters

in	*Path*	path to the object
in	*Mode*	New access to the object

Return values

0	No error. Mode altered as directed.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

9.6.2.6 int hpss_Chown (const char ∗ *Path,* uid_t *Owner,* gid_t *Group*)

Changes the user id and group id of a file or directory.

The 'hpss_Chown' function sets the user id and the group id of the file or directory named by *Path* to the values specified by *Owner* and *Group*, respectively. It sets the account id appropriately with respect to the new uid.

Parameters

in	*Path*	path to the object
in	*Owner*	desired new owner ID
in	*Group*	desired new value for the group owner

Return values

0	No error. Changes made correctly.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation.

9.6.2.7 int hpss_FileGetAttributes (const char ∗ *Path,* hpss_fileattr_t ∗ *AttrOut*)

Queries the attributes of a file.

The hpss_FileGetAttributes function can be used to query attributes on an entry in the name/file system that is refered to by *Path*.

Parameters

in	*Path*	Path to the object
out	*AttrOut*	Object attributes

Return values

0	No error, caller has access.

-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path or AttrOut parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string..
-ENOTDIR	A component of the Path prefix is not a directory.

Note

This function reads through symbolic links and junctions. It cannot be used to get the attributes of the symbolic link or junction itself.
The timestamp and permissions contained within the HPSS attribute structure are not POSIX; they must be converted using the appropriate HPSS conversion functions.

9.6.2.8 int hpss_FileGetAttributesBitfile (const bfs_bitfile_obj_handle_t ∗ *BitfileObj,* char ∗ *Path,* hpss_fileattr_t ∗ *AttrOut*)

Get file attributes using a Bitfile Object.

This function can be used to query attributes on an entry in the name/file system that is refered to by 'Bitfile'. It returns one of possibly multiple path names to the bitfile and the file attributes structure for the bitfile.

Parameters

`in`	*BitfileObj*	Bitfile Object
`out`	*Path*	Path to the object
`out`	*AttrOut*	Object Attributes

Return values

0	No error, caller has access.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path or AttrOut parameter is a NULL pointer.
-EINVAL	BitfileObj is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

Note

The timestamp and permissions contained within the HPSS attribute structure are not POSIX; they must be converted using the appropriate HPSS conversion functions.
A Bitfile may have multiple paths due to symlinks, junctions, etc. The path returned may be just one of several paths to the Bitfile. The path returned begins from the last junction that was crossed. The remaining path back to the absolute root can be obtained by using looking up the junction's path name in the file attributes, potentially recursively, back to the HPSS FILESET_ROOT.

9.6.2.9 int hpss_FileGetAttributesHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* hpss_fileattr_t ∗ *AttrOut*)

Queries the attributes of a file using a handle.

The hpss_FileGetAttributesHandle function can be used to query attributes on an entry in the name/file system that is referred to by *ObjHandle* and *Path*.

Parameters

`in`	*ObjHandle*	parent object handle
`in`	*Path*	path to the object
`in`	*Ucred*	user credentials
`out`	*AttrOut*	attributes after query

Return values

0	No error, caller has access.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path or AttrOut parameter is a NULL pointer.
-EINVAL	ObjHandle is NULL.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

Note

This call allocates memory for the returned physical volume conformant array that must be freed by the caller. The following code is an example of how to free this memory.
The timestamp and permissions contained in the attributes which are returned by this function are not POSIX. They should be converted using an appropriate HPSS conversion function.

9.6.2.10 int hpss_FileGetXAttributes (const char ∗ *Path,* uint32_t *Flags,* uint32_t *StorageLevel,* hpss_xfileattr_t ∗ *AttrOut*)

Queries the attributes of a file using flags and storage level perms.

The hpss_FileGetXAttributes function can be used to query attributes on an entry in the name/file system that is refered to by *Path*. Additional Flags and StorageLevel parms allow getting storage attrs. If the Class of Service uses Stage on Open, or if the bitfile metadata is otherwise locked for some long period of time, the call will block until the blocking operation is completed unless the API_GET_XATTRS_NO_BLOCK flag is provided.

Parameters

`in`	*Path*	Path to the object
`in`	*Flags*	Flags for storage attrs • API_GET_STATS_FOR_LEVEL - Returns bitfile attributes at the storage level specified by the *StorageLevel* argument • API_GET_STATS_FOR_1STLEVEL - Returns bitfile attributes at th first storage level whether or not it contains bitfile data. • API_GET_STATS_FOR_OPTIMIZE - Returns only StripeWidth and OptimumAccessSize for storage level zero • API_GET_STATS_FOR_ALL_LEVELS - Returns bitfile attributes across all storage levels • API_GET_XATTRS_NO_BLOCK - Causes the operation to be non-blocking, however this data may be inconsistent with metadata.

in	*StorageLevel*	Storage level to query
out	*AttrOut*	Object Attributes

Return values

0	No error, caller has access.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path or AttrOut parameter is a NULL pointer.
-EINVAL	BitfileID is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

Note

This call allocates memory for the returned physical volume conformant array that must be freed by the caller. The following code is an example of how to free this memory.

```
for(i=0;i<HPSS_MAX_STORAGE_LEVELS;i++)
{
   for(j=0;j<AttrOut.SCAttrib[i].NumberOfVVs;j++)
   {
      if (AttrOut.SCAttrib[i].VVAttrib[j].PVList != NULL)
      {
         free(AttrOut.SCAttrib[i].VVAttrib[j].PVList);
      }
   }
}
```

Note

This function reads through symbolic links and junctions and cannot be used to get the attributes of symbolic links or junctions.
The timestamp and permissions contained in the attributes which are returned by this function are not POSIX. They should be converted using an appropriate HPSS conversion function.

9.6.2.11 int hpss_FileGetXAttributesHandle (const **ns_ObjHandle_t** ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* uint32_t *Flags,* uint32_t *StorageLevel,* **hpss_xfileattr_t** ∗ *AttrOut*)

Queries the attributes of a file using a handle, flags, and a storage level.

The hpss_FileGetXAttributesHandle function can be used to query attributes on an entry in the name/file system that is referred to by 'ObjHandle' and 'Path'.

Parameters

in	*ObjHandle*	parent object handle
in	*Path*	path to the object
in	*Ucred*	user credentials
in	*Flags*	flags for storage attrs
in	*StorageLevel*	level to query
out	*AttrOut*	attributes after query

Return values

0	No error, caller has access.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path or AttrOut parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

Warning

The storage level information must be freed by the application.

9.6.2.12 int hpss_FileSetAttributes (const char ∗ *Path,* hpss_fileattrbits_t *SelFlags,* const hpss_fileattr_t ∗ *AttrIn,* hpss_fileattr_t ∗ *AttrOut*)

Changes the attributes of an object.

The hpss_FileSetAttributes function can be used to change attributes on an entry in the name/file system that is refered to by 'Path'.

Parameters

`in`	*Path*	path to the object
`in`	*SelFlags*	Attributes fields to set; the following is a list of file attributes which can be set with this function: • CORE_ATTR_ACCOUNT • CORE_ATTR_COMMENT • CORE_ATTR_COMPOSITE_PERMS • CORE_ATTR_COS_ID • CORE_ATTR_DATA_LENGTH • CORE_ATTR_DM_DATA_STATE_FLAGS • CORE_ATTR_DM_HANDLE • CORE_ATTR_DM_HANDLE_LENGTHCORE_ATTR_DONT_PURGE • CORE_ATTR_ENTRY_COUNT • CORE_ATTR_EXTENDED_ACLS • CORE_ATTR_FAMILY_ID • CORE_ATTR_FILESET_HANDLE • CORE_ATTR_FILESET_ID • CORE_ATTR_FILESET_ROOT_ID • CORE_ATTR_FILESET_STATE_FLAGS • CORE_ATTR_FILESET_TYPE • CORE_ATTR_GATEWAY_UUID • CORE_ATTR_GIDCORE_ATTR_GROUP_PERMS • CORE_ATTR_LINK_COUNT • CORE_ATTR_MAC_SEC_LABEL • CORE_ATTR_OPEN_COUNT • CORE_ATTR_OTHER_PERMS • CORE_ATTR_READ_COUNT • CORE_ATTR_REALM_ID • CORE_ATTR_REGISTER_BITMAP • CORE_ATTR_SET_GIDCORE_ATTR_SET_STICKY • CORE_ATTR_SET_UID • CORE_ATTR_SUB_SYSTEM_ID • CORE_ATTR_TIME_CREATED • CORE_ATTR_TIME_LAST_READ • CORE_ATTR_TIME_LAST_WRITTEN • CORE_ATTR_TIME_MODIFIED • CORE_ATTR_TYPE • CORE_ATTR_UID • CORE_ATTR_USER_PERMS • CORE_ATTR_WRITE_COUNT

out	*AttrOut*	attributes after change

Return values

0	No error, caller has access.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path, AttrIn or AttrOut parameter is a NULL pointer.
-EINVAL	An attribute value or selection flag is invalid.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOSPC	Resources could not be allocated to satisfy the request.
-ENOTDIR	A component of the Path prefix is not a directory.
-EOPNOTSUPP	The requested change is not supported.
-EPERM	The client does not have the appropriate privileges to change the file's attributes.

Note

The Bitfile ID cannot be set using this function.
The regular purge lock and super purge lock cannot be set using this function. They are set using the hpss_-PurgeLock function.
The Account Code can be set only if site-style accounting is being used.
The timestamp and permissions contained in the attributes which are returned by this function are not POSIX. They should be converted using an appropriate HPSS conversion function.

9.6.2.13 int hpss_FileSetAttributesBitfile (const **bfs_bitfile_obj_handle_t** ∗ *BitfileObj,* **hpss_fileattrbits_t** *SelFlags,* const **hpss_fileattr_t** ∗ *AttrIn,* **hpss_fileattr_t** ∗ *AttrOut,* char ∗ *Path*)

Change the attributes of an object specified by bitfile object.

The hpss_FileSetAttributesBitfile function can be used to set attributes on an entry in the name/file system that is refered to by 'BitfileID'. It returns one of possibly multiple path names to the bitfile and the file attributes structure for the entry.

Parameters

in	*BitfileObj*	Bitfile Object
in	*SelFlags*	attributes fields to set; refer to hpss_SetFileAttributes for a list
in	*AttrIn*	input attributes
out	*AttrOut*	attributes after query
out	*Path*	path to the object

Return values

0	No error, caller has access.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The AttrOut parameter is a NULL pointer.
-EINVAL	An attribute value or selection flag is invalid, or BitfileObj is NULL.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOSPC	Resources could not be allocated to satisfy the request.

Note

The Bitfile ID cannot be set using this function.
The regular purge lock and super purge lock cannot be set using this function. They are set using the hpss_-PurgeLock function.
The Account Code can be set only if site-style accounting is being used.
The timestamp and permissions contained in the attributes which are returned by this function are not POSIX. They should be converted using an appropriate HPSS conversion function.

9.6.2.14 int hpss_FileSetAttributesHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* hpss_fileattrbits_t *SelFlags,* const hpss_fileattr_t ∗ *AttrIn,* hpss_fileattr_t ∗ *AttrOut*)

Changes the attributes of an object based upon an object handle and path.

The hpss_FileSetAttributesHandle function can be used to change attributes on an entry in the name/file system that is referred to by 'Path' and 'ObjHandle'.

Parameters

in	*ObjHandle*	parent object handle
in	*Path*	path to the object
in	*Ucred*	user credentials
in	*SelFlags*	attributes fields to set; refer to hpss_SetFileAttributes for a list
in	*AttrIn*	input attributes
out	*AttrOut*	attributes after change

Return values

0	No error, caller has access.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path, AttrIn or AttrOut parameter is a NULL pointer.
-EINVAL	An attribute value or selection flag is invalid, or the ObjHandle is NULL.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOSPC	Resources could not be allocated to satisfy the request.
-ENOTDIR	A component of the Path prefix is not a directory.
-EOPNOTSUPP	The requested change is not supported.
-EPERM	The client does not have the appropriate privileges to change the file's attributes.

Note

The Bitfile ID cannot be set using this function.
The regular purge lock and super purge lock cannot be set using this function. They are set using the hpss_-PurgeLock function.
The Account Code can be set only if site-style accounting is being used.
The timestamp and permissions contained in the attributes which are returned by this function are not POSIX. They should be converted using an appropriate HPSS conversion function.

9.6.2.15 int hpss_FileSetCOS (const char ∗ *Path,* uint32_t *COSId,* uint32_t *StreamId*)

Change the COS of an file.

The hpss_FileSetCOS function can be used to change the COS on an entry in the name/file system that is refered to by *Path*. It can also be used to assign the COS change to a specific COS change thread in the Core Server.

Parameters

in	*Path*	path to the object
in	*COSId*	change COS to this value
in	*StreamId*	have Core Server use this stream

Return values

0	No error, caller has access.
-ENOMEM	Not Enough Memory
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	An attribute value or selection flag is invalid.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOSPC	Resources could not be allocated to satisfy the request.
-ENOTDIR	A component of the Path prefix is not a directory.
-EOPNOTSUPP	The requested change is not supported.
-EPERM	The client does not have the appropriate privileges to change the file's attributes.

9.6.2.16 int hpss_FileSetCOSHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* uint32_t *COSId,* uint32_t *StreamId*)

Change the COS of a file.

The hpss_FileSetCOSHandle function can be used to change the COS on an entry in the name/file system that is refered to by *Path* and *ObjHandle*. It can also provide a *StreamId* to control which Core Server COS change thread is assigned the operation

Parameters

in	*ObjHandle*	parent object handle
in	*Path*	path to the object
in	*Ucred*	user credentials
in	*COSId*	change COS to this value
in	*StreamId*	used this Core Server stream

Return values

0	No error, caller has access.
-ENOMEM	Not Enough Memory
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	An attribute value or selection flag is invalid.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOSPC	Resources could not be allocated to satisfy the request.
-ENOTDIR	A component of the Path prefix is not a directory.
-EOPNOTSUPP	The requested change is not supported.
-EPERM	The client does not have the appropriate privileges to change the file's attributes.

9.6.2.17 int hpss_Fstat (int *Fildes*, hpss_stat_t ∗ *Buf*)

Retrieve statistics for an open file.

The 'hpss_Fstat' function obtains information about the open file or directory specified by *Fildes* and returns the information in a structure pointed to by *Buf*.

Parameters

in	*Fildes*	ID of open file/directory
out	*Buf*	Returned statistics

Return values

0	No error. Statistics in buffer.
-EBADF	The file descriptor supplied does not correspond to an open file.
-EBUSY	Another thread is currently manipulating this entry.
-EFAULT	The Buf parameter is a NULL pointer or no object handle is present.

Note

hpss_OpenBitfileVAttrs provides a cached NS object handle. This function uses that object handle in order to obtain information, so if the cached object handle is invalid or belongs to a different file, errors or invalid information may be returned.
Using this API with a Fildes corresponding to a file opened via hpss_ReopenBitfile() will cause this function to return an error; use hpss_OpenBitfile() instead.
This function cannot be used to get attributes for a file which has been unlinked.

9.6.2.18 int hpss_GetAttrHandle (const ns_ObjHandle_t ∗ *ObjHandle*, const char ∗ *Path*, const sec_cred_t ∗ *Ucred*, ns_ObjHandle_t ∗ *HandleOut*, hpss_vattr_t ∗ *AttrOut*)

Obtains information about a file or directory using a handle.

The 'hpss_GetAttrHandle' function obtains information about the file or directory named by *Path*, taken relative to the directory indicated by *ObjHandle*. Attributes are returned in the area pointed to by *AttrOut*. If *Path* refers to a symbolic link, information will be returned about the link itself.

Parameters

in	*ObjHandle*	Parent object handle
in	*Path*	Path of file to get attributes
in	*Ucred*	User credentials
out	*HandleOut*	Returned object handle for *Path*
out	*AttrOut*	Returned attributes for *Path*

Return values

0	No error. Valid information returned.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path or AttrOut parameter is a NULL pointer.
-EINVAL	The ObjHandle parameter is NULL.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

9.6.2.19 int hpss_GetFileNotrunc (const char ∗ *Path,* int ∗ *NotruncFlag*)

Retrieves the file's NOTRUNC_FINAL_SEG flag.

The 'hpss_GetFileNotrunc' function returns the value of the NOTRUNC_FINAL_SEG flag in the bitfile descriptor. Unless this flag is set, the final segment of the file will be truncated to best fit the data when the file is closed.

Parameters

in	*Path*	Pathname of file
out	*NotruncFlag*	Value of NOTRUNC_FINAL_SEG flag

Return values

0	Success
-EINVAL	Path is not a regular file.
-ENOENT	Path is an empty string or does not exist.
-EFAULT	Path or NotruncFlag is a NULL pointer.

9.6.2.20 int hpss_GetListAttrs (const char ∗ *Path,* hpss_Attrs_t ∗ *AttrOut*)

Retrieve file statistics associated with directory listing operations.

The 'hpss_GetListAttrs' function obtains information needed to support directory listing operations for the file or directory named by 'Path' and returns the information in a structure pointed to by 'AttrOut'. If 'Path' refers to a junction, the junction will be traversed to get to the target object. If 'Path' refers to a symbolic link, information will be returned about the link itself rather than the target object.

Parameters

in	*Path*	path of file to get statistics
out	*AttrOut*	Returned statistics

Return values

0	No error. Statistics in buffer.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path or AttrOut parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the systemimposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

9.6.2.21 int hpss_GetRawAttrHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* ns_ObjHandle_t ∗ *HandleOut,* hpss_vattr_t ∗ *AttrOut*)

Obtains information about a symlink or junction using a handle.

The 'hpss_GetRawAttrHandle' function obtains information about the symlink or the junction named by 'Path', taken relative to the directory indicated by 'ObjHandle'. Attributes are returned in the area pointed to by 'AttrOut'.

Parameters

`in`	*ObjHandle*	Parent object handle
`in`	*Path*	Path of file to get attributes
`in`	*Ucred*	User credentials
`out`	*HandleOut*	Returned object handle for *Path*
`out`	*AttrOut*	Returned attributes for *Path*

Return values

0	No error. Valid information returned.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path or AttrOut parameter is a NULL pointer.
-EINVAL	The ObjHandle parameter is NULL.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

9.6.2.22 int hpss_Lstat (const char ∗ *Path,* hpss_stat_t ∗ *Buf*)

Retrieve file, or link, statistics.

The 'hpss_Lstat' function obtains information about the file or directory named by 'Path' and returns the information in a structure pointed to by 'Buf'. If 'Path' refers to a junction, the junction will be traversed to get to the target object. If 'Path' refers to a symbolic link, information will be returned about the link itself.

Parameters

`in`	*Path*	path of file to get statistics
`out`	*Buf*	Returned statistics

Return values

0	No error. Statistics in buffer.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path or Buf parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

9.6.2.23 int hpss_PurgeLock (int *Fildes,* purgelock_flag_t *Flag*)

Locks or unlocks a file's purge state.

The 'hpss_PurgeLock' function locks/unlocks the purge state of the file associated with the handle, 'Fildes'. The regular purge lock locks the file to the top of its hierarchy if that storage level is disk. The super purge lock locks the file at all levels regardless of whether they are disk or tape. Only an authorized caller (one who has control permission on the core server ACL) may set or clear the super purge lock.

Parameters

in	*Fildes*	ID of open object
in	*Flag*	Purge Lock or Unlock, Regular or Super Purge • PURGE_LOCK - Set the regular purge lock. • PURGE_UNLOCK - Clear the regular purge lock. • SUPER_PURGE_LOCK - Set the super purge lock. • SUPER_PURGE_UNLOCK - Clear the super purge lock.

Return values

0	Success
-EBADF	The supplied file descriptor does not correspond to an open file.
-EBUSY	The specified file descriptor is in use.
-ESTALE	The connection for this entry is not valid.
-EINVAL	An incorrect value was specified for Flag.

9.6.2.24 int hpss_PurgeOnMigrate (int *Fildes*, purgeonmigrate_flag_t *Flag*)

Sets or clears the purge on migrate flag.

The 'hpss_PurgeOnMigrate' function sets or clears the purge on migrate flag.

Parameters

in	*Fildes*	Open file handle
in	*Flag*	Clear or set the purge on migrate setting. • PURGE_ON_MIGRATE_CLEAR - Clear the purge on migrate setting. • PURGE_ON_MIGRATE_SET - Set the purge on migrate setting.

Return values

0	Success
-EBADF	The supplied file descriptor does not correspond to an open file.
-EBUSY	The specified file descriptor is in use.
-ESTALE	The connection for this entry is not valid.
-EINVAL	An incorrect value was specified for Flag.

9.6.2.25 int hpss_SetAttrHandle (const ns_ObjHandle_t ∗ *ObjHandle*, const char ∗ *Path*, const sec_cred_t ∗ *Ucred*, uint64_t *SelFlagsIn*, const hpss_vattr_t ∗ *AttrIn*, uint64_t ∗ *SelFlagsOut*, hpss_vattr_t ∗ *AttrOut*)

Updates attribute values for a file or directory.

The 'hpss_SetAttrHandle' function updates attribute values for the file or directory named by 'Path', taken relative to the directory indicated by 'ObjHandle'. The attributes to be updated are indicated by 'SelFlagsIn', and the new attribute values are contained in the structure pointed to by 'AttrIn'. Indication of attributes actually changed are returned in 'SelFlagsOut', and the new attribute values in 'AttrOut'. If the input Path refers to a symbolic link, hpss_-SetAttrHandle returns information about the link itself. The hpss_FileSet∗ attributes functions operate on the object to which the link points.

Parameters

in	*ObjHandle*	parent object handle
in	*Path*	path of file to set attributes
in	*Ucred*	user credentials
in	*SelFlagsIn*	indication of attrs to be set
in	*AttrIn*	new attribute values
out	*SelFlagsOut*	indication of attrs set
out	*AttrOut*	returned attributes

Return values

0	No error.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path, AttrIn or AttrOut parameter is a NULL pointer.
-EINVAL	An attribute value or selection flag is invalid, or the ObjHandle is NULL.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOSPC	Resources could not be allocated to satisfy the request.
-ENOTDIR	A component of the Path prefix is not a directory.
-EOPNOTSUPP	The requested change is not supported.
-EPERM	The client does not have the appropriate privileges to change the file's attributes.

Note

The regular purge lock and super purge lock cannot be set using this function. They are set using the hpss_-PurgeLock function.

9.6.2.26 int hpss_SetCOSByHints (int *Fildes,* uint32_t *Flags,* const hpss_cos_hints_t ∗ *HintsPtr,* const hpss_cos_priorities_t ∗ *PrioPtr,* hpss_cos_md_t ∗ *COSPtr*)

Sets the COS of an empty file.

The 'hpss_SetCOSByHints' function attempts to set the COS of a previously open, empty file.

Parameters

in	*Fildes*	ID of object to be read
in	*Flags*	operation flags to use
in	*HintsPtr*	COS hints desired
in	*PrioPtr*	Priorities of COS hints
out	*COSPtr*	COS used

Return values

0	No error occurred
-EBADF	The specified file descriptor does not refer to an open file.
-EBUSY	The file is currently in use by another client thread, or the Core Server could not complete the request at the current time.
-EFAULT	One of HinstPtr, PrioPtr, or COSPtr is a NULL pointer.
-EINVAL	The specified COS hints are invalid.

-EPERM	The client does not have the appropriate privileges to perform the request.
-ENOTDIR	A component of the Path prefix is not a directory.

Note

This is currently used when performing the first write of an empty file in an attempt to put the file in an appropriate COS based on some limited knowledge of the eventual file size.

9.6.2.27 int hpss_SetFileNotrunc (const char ∗ *Path,* notrunc_flag_t *Flag*)

Sets or clears a file's NOTRUNC_FINAL_SEG flag.

The 'hpss_SetFileNotrunc' function sets or clears the NOTRUNC_FINAL_SEG flag in the bitfile descriptor. Unless this flag is set, the final segment of the file will be truncated to best fit the data when the file is closed. The user must be the file owner or root to set the flag. Only authorized callers can clear the flag. These rules are enforced by the core server.

Parameters

`in`	*Path*	Pathname of file
`in`	*Flag*	Whether to set or clear flag

Return values

0	Success.
-EINVAL	Path is not a regular file.
-ENOENT	Path is an empty string or does not exist.
-EFAULT	Path is a NULL pointer.
-EPERM	User does not have necessary privileges.

9.6.2.28 int hpss_Stat (const char ∗ *Path,* hpss_stat_t ∗ *Buf*)

Retrieve file statistics.

The 'hpss_Stat' function obtains information about the file or directory named by 'Path' and returns the information in a structure pointed to by 'Buf'. If 'Path' refers to a junction or symbolic link, the junction or link will be traversed to get to the target object.

Parameters

`in`	*Path*	path of file to get statistics
`out`	*Buf*	Returned statistics

Return values

0	No error. Statistics in buffer.
-EACCES	Search permission is denied for a component of the path prefix.
-EFAULT	The Path or Buf parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

9.6.2.29 int hpss_Utime (const char * *Path,* const struct utimbuf * *Times*)

Set the access and modification times of a file.

The 'hpss_Utime' function set the access and modification times of the file 'Path' to the values specified in the structure pointed to by 'Times'.

Parameters

`in`	*Path*	path of the object
`in`	*Times*	new time values

Return values

0	Success
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation.

9.7 File Name

Functions used to rename or remove names associated with files.

Functions

- int hpss_Link (const char ∗Existing, const char ∗New)

  Creates a link to a file.
- int hpss_LinkHandle (const ns_ObjHandle_t ∗ObjHandle, const ns_ObjHandle_t ∗DirHandle, const char ∗New, const sec_cred_t ∗Ucred)

  Creates a link.
- int hpss_LinkHandleParent (const ns_ObjHandle_t ∗SrcDirHandle, const char ∗SrcFile, const ns_ObjHandle-_t ∗DestDirHandle, const char ∗DestFile, const sec_cred_t ∗Ucred)

  Creates a link.
- int hpss_Rename (const char ∗Old, const char ∗New)

  Rename an HPSS file or directory.
- int hpss_RenameHandle (const ns_ObjHandle_t ∗OldHandle, const char ∗OldPath, const ns_ObjHandle_t ∗NewHandle, const char ∗NewPath, const sec_cred_t ∗Ucred)

  Rename an HPSS file or directory using an object handle.
- int hpss_Symlink (const char ∗Contents, const char ∗Path)

  Create a symbolic link.
- int hpss_SymlinkHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Contents, const char ∗Path, const sec_cred_t ∗Ucred, hpss_vattr_t ∗AttrsOut)

  Create a symbolic link using a handle.

9.7.1 Detailed Description

Functions used to rename or remove names associated with files.

9.7.2 Function Documentation

9.7.2.1 int hpss_Link (const char ∗ *Existing,* const char ∗ *New*)

Creates a link to a file.

The ’hpss_Link’ function creates a new link to the file currently named by ’Existing’, with the name specified by ’New’.

Parameters

in	*Existing*	Existing name of the object
in	*New*	New name of the object

Return values

0	Link was successful.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the new link.

-EEXIST	The object identified by New already exists.
-EFAULT	The Existing or New parameter is a NULL pointer.
-EPERM	The object specified by Existing is a directory.
-EMLINK	The object specified by Existing is a directory.
-ENAMETOOLONG	The length of the Existing or New argument exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
-ENOENT	No entry exists for the specified file.
-ENOTDIR	A component of the path prefix is not a directory.

9.7.2.2 int hpss_LinkHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const ns_ObjHandle_t ∗ *DirHandle,* const char ∗ *New,* const sec_cred_t ∗ *Ucred*)

Creates a link.

The ’hpss_LinkHandle’ function creates a new link to the file associated with ’ObjHandle’, with the name ’New’, taken relative to the directory specified by ’DirHandle’.

Parameters

`in`	*ObjHandle*	handle of existing obj
`in`	*DirHandle*	handle of parent directory
`in`	*New*	New name of the object
`in`	*Ucred*	user credentials

Return values

0	Link was successful.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the new link.
-EEXIST	The object identified by New already exists.
-EFAULT	The ObjHandle, DirHandle, or New parameter is a NULL pointer.
-EPERM	The object specified by ObjHandle is a directory.
-EMLINK	The number of links to the file named by ObjHandle would exceed the system-imposed limit.
-ENAMETOOLONG	The length of the New argument exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
-ENOENT	New points to a null string.
-ENOTDIR	A component of the path prefix is not a directory.

9.7.2.3 int hpss_LinkHandleParent (const ns_ObjHandle_t ∗ *SrcDirHandle,* const char ∗ *SrcFile,* const ns_ObjHandle_t ∗ *DestDirHandle,* const char ∗ *DestFile,* const sec_cred_t ∗ *Ucred*)

Creates a link.

This function creates a new link to the file associated with ’SrcObjHandle’ and ’SrcPath’, with the name ’New’, taken relative to the directory specified by ’DirHandle’.

Parameters

`in`	*SrcDirHandle*	Handle to source parent directory
`in`	*SrcFile*	Path within source parent directory
`in`	*DestDirHandle*	Handle to destination parent directory

in	*DestFile*	New link path within destination parent directory
in	*Ucred*	user credentials

Return values

0	Link was successful.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the new link.
-EEXIST	The object identified by DestObjHandle and DestPath already exists.
-EFAULT	The SrcDirHandle, DestDirHandle, SrcFile, or DestFile parameter is a NULL pointer.
-EPERM	The object specified by SrcDirHandle and SrcFile is a directory.
-EMLINK	The number of links to the source object would exceed the system-imposed limit.
-ENAMETOOLONG	The length of the DestFile argument exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
-ENOENT	New points to a null string.
-ENOTDIR	A component of the path prefix is not a directory.

Note

This API differs from hpss_LinkHandle() in that only the parent handles, not the handle to the source object, must be provided. This can be useful in cases where the goal is to minimize path lookups at a directory level.

9.7.2.4 int hpss_Rename (const char ∗ *Old,* const char ∗ *New*)

Rename an HPSS file or directory.

The 'hpss_Rename' function changes the name of the file or directory currently named by 'Old', to the new name 'New'.

Parameters

in	*Old*	Old name of the object
in	*New*	New name of the object

Return values

0	Rename was successful.
-EACCES	Search permission is denied on a component of the path prefix, or one of the directories containing Old or New denies write permission, or write permission is required and denied for a directory pointed to by the Old or New arguments.
-EFAULT	The Old or New parameter is a NULL pointer.
-EISDIR	The New argument points to a directory, and the Old argument points to a file that is not a directory.
-EMLINK	The file named by Old is a directory, and the link count of the parent directory of New already contains the maximum allowed number of links.
-ENAMETOOLONG	The length of the Old or New argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Old or New argument points to an empty string.
-ENOTDIR	A component of the path prefix is not a directory.
-ENOTEMPTY	The path named by New is a directory containing entries other than dot and dot-dot.

Note

If New already exists and would be overwritten by the rename, and trashcans are enabled, the existing New is moved to the trashcan corresponding to to the deleting user and its location.

9.7.2.5 int hpss_RenameHandle (const ns_ObjHandle_t ∗ *OldHandle,* const char ∗ *OldPath,* const ns_ObjHandle_t ∗ *NewHandle,* const char ∗ *NewPath,* const sec_cred_t ∗ *Ucred*)

Rename an HPSS file or directory using an object handle.

The 'hpss_RenameHandle' function changes the name of the file or directory currently named by 'OldPath', taken relative to the directory indicated by 'OldHandle' to the new name 'NewPath', taken relative to the directory indicated by 'NewHandle'.

Parameters

`in`	*OldHandle*	Old parent object handle
`in`	*OldPath*	Old name of the object
`in`	*NewHandle*	New parent object handle
`in`	*NewPath*	New name of the object
`in`	*Ucred*	User credentials

Return values

0	Rename was successful.
-EACCES	Search permission is denied on a component of the path prefix, or one of the directories containing OldHandle or NewHandle denies write permission, or write permission is required and denied for a directory pointed to by the OldHandle or NewHandle arguments.
-EFAULT	NewPath is a NULL pointer.
-EINVAL	The NewHandle or OldHandle parameter is NULL.
-EISDIR	The NewHandle argument points to a directory, and the OldHandle argument points to a file that is not a directory.
-EMLINK	The file named by OldHandle is a directory, and the link count of the parent directory of NewHandle already contains the maximum allowed number of links.
-ENAMETOOLONG	The length of OldPath or NewPath exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the OldPath or NewPath argument points to an empty string.
-ENOTDIR	A component of the path prefix is not a directory.
-ENOTEMPTY	The path named by NewPath is a directory containing entries other than dot and dot dot.

Note

If New already exists and would be overwritten by the rename, and trashcans are enabled, the existing New is moved to the trashcan corresponding to to the deleting user and its location.

9.7.2.6 int hpss_Symlink (const char ∗ *Contents,* const char ∗ *Path*)

Create a symbolic link.

The 'hpss_Symlink' function creates a symbolic link, named by 'Path', with contents specified by 'Contents'.

Parameters

in	*Contents*	Desired contents of the new link
in	*Path*	Name of the link

Return values

0	Creation of symlink was successful.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the specified path name.
-EFAULT	The Path or Contents parameter is a NULL pointer.
-EEXIST	The specified link already exists.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	No entry exists for a component of the path name.
-ENOTDIR	A component of the Path prefix is not a directory.

9.7.2.7 int hpss_SymlinkHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Contents,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* hpss_vattr_t ∗ *AttrsOut*)

Create a symbolic link using a handle.

The 'hpss_SymlinkHandle' function creates a symbolic link with the name 'Path' (taken relative to 'ObjHandle'), with the contents specified by 'Contents'.

Parameters

in	*ObjHandle*	handle of parent
in	*Contents*	Contents of the link
in	*Path*	New name of symlink
in	*Ucred*	user credentials
out	*AttrsOut*	symbolic link attributes

Return values

0	Creation of symbolic link was successful.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the specified path name.
-EFAULT	The Path or Contents parameter is a NULL pointer.
-EEXIST	The specified link already exists.
-EINVAL	ObjHandle is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

9.8 ACLs

Functions used to create, remove, and manipulate ACLs on files or directories.

Functions

- int hpss_DeleteACL (const char ∗Path, const uint32_t Options, const ns_ACLConfArray_t ∗ACL)

    Removes an ACL entry from a file or directory.
- int hpss_DeleteACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, const uint32_t Options, const ns_ACLConfArray_t ∗ACL)

    Removes an ACL entry from a file or directory using a handle.
- int hpss_GetACL (const char ∗Path, uint32_t Options, ns_ACLConfArray_t ∗ACL)

    Retrives the ACL of a file or directory.
- int hpss_GetACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, const uint32_t Options, ns_ACLConfArray_t ∗ACL)

    Retrieves an ACL of a file or directory using a handle.
- int hpss_SetACL (const char ∗Path, const uint32_t Options, const ns_ACLConfArray_t ∗ACL)

    Sets the ACL entries of a file or directory.
- int hpss_SetACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, const uint32_t Options, const ns_ACLConfArray_t ∗ACL)

    Sets the ACL entries of a file or directory using a handle.
- int hpss_UpdateACL (const char ∗Path, int32_t Options, const ns_ACLConfArray_t ∗ACL)

    Updates an ACL array of a file or directory.
- int hpss_UpdateACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t Options, const ns_ACLConfArray_t ∗ACL)

    Updates an ACL entry of a file or directory using a handle.

9.8.1 Detailed Description

Functions used to create, remove, and manipulate ACLs on files or directories.

9.8.2 Function Documentation

9.8.2.1 int hpss_DeleteACL (const char ∗ *Path,* const uint32_t *Options,* const ns_ACLConfArray_t ∗ *ACL*)

Removes an ACL entry from a file or directory.

The 'hpss_DeleteACL' function removes an ACL entry from the Access Control List of a file or directory named by *Path*.

Parameters

`in`	*Path*	Names the file for which the *ACL* is being removed.
`in`	*Options*	Bit vector used to specify what type of ACL is to be removed. One of: • - *HPSS_ACL_OPTION_OBJ* Normal ACL • - *HPSS_ACL_OPTION_IO* Initial Object ACL (only valid for directory objects) • - *HPSS_ACL_OPTION_IC* Initial Container ACL (only valid for directory objects)
`in`	*ACL*	Points to the list of ACL entries to be removed.

Return values

0	No error.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path or ACL parameter is a NULL parameter.
-EINVAL	Exactly one of the HPSS_ACL_OPTION_∗ bits must be set in the Options bit vector.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation.

See Also

hpss_GetACL hpss_UpdateACL hpss_SetACL

9.8.2.2 int hpss_DeleteACLHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* const uint32_t *Options,* const ns_ACLConfArray_t ∗ *ACL*)

Removes an ACL entry from a file or directory using a handle.

The 'hpss_DeleteACLHandle' function removes an ACL entry from the Access Control List of a file or directory named by *Path*, taken relative to the directory indicated by 'ObjHandle'.

Parameters

`in`	*ObjHandle*	Parent object handle
`in`	*Path*	File for which the ACL is being removed
`in`	*Ucred*	User credentials
`in`	*Options*	Bit vector used to specify what type of ACL is to be removed. One of: • - *HPSS_ACL_OPTION_OBJ* Normal ACL • - *HPSS_ACL_OPTION_IO* Initial Object ACL (only valid for directory objects) • - *HPSS_ACL_OPTION_IC* Initial Container ACL (only valid for directory objects)
`in`	*ACL*	Points to the list of ACL entries to be removed.

Return values

0	No error.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path or ACL parameter is a NULL pointer.
-EINVAL	Exactly one of the HPSS_ACL_OPTION_∗ bits must be set in the Options bit vector.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

-EPERM	The client does not have the appropriate privileges to perform the operation.
-ESRCH	A specified ACL entry did not match an existing ACL entry for the file.

See Also

hpss_GetACLHandle hpss_UpdateACLHandle hpss_SetACLHandle

9.8.2.3 int hpss_GetACL (const char ∗ *Path,* uint32_t *Options,* ns_ACLConfArray_t ∗ *ACL*)

Retrives the ACL of a file or directory.

The 'hpss_GetACL' function queries the Access Control List of the file or directory named by 'Path'.

Parameters

`in`	*Path*	path of file
`in`	*Options*	bitmask of ACL options HPSS_ACL_OPTION_OBJ - return the object's normal ACL HPSS_ACL_OPTION_IO - return the initial-object ACL (only valid for directory objects) HPSS_ACL_OPTION_IC - return the initial-container ACL (only valid for directory objects)
`out`	*ACL*	ACL array for the file/dir

Return values

0	No error. ACL in buffer.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path or ACL parameter is a NULL pointer.
-EINVAL	Exactly one of the HPSS_ACL_OPTION_∗ bits must be set in the Options bit vector.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation.

Note

The caller is responsible for freeing the ACL return parameter.

9.8.2.4 int hpss_GetACLHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* const uint32_t *Options,* ns_ACLConfArray_t ∗ *ACL*)

Retrieves an ACL of a file or directory using a handle.

The 'hpss_GetACLHandle' function queries the Access Control List of a file or directory named by 'Path', taken relative to the directory indicated by 'ObjHandle'.

Parameters

`in`	*ObjHandle*	parent object handle
`in`	*Path*	path of file

in	*Ucred*	user credentials
in	*Options*	bitmask of ACL options HPSS_ACL_OPTION_OBJ - return the object's normal ACL HPSS_ACL_OPTION_IO - return the initial-object ACL (only valid for directory objects) HPSS_ACL_OPTION_IC - return the initial-container ACL (only valid for directory objects)
out	*ACL*	returned ACL array

Return values

0	No error.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path or ACL parameter is a NULL pointer.
-EINVAL	Exactly one of the HPSS_ACL_OPTION_∗ bits must be set in the Options bit vector.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation.

Note

The caller is responsible for freeing the ACL return parameter.
If the Ucred parameter is NULL, the credentials in the current thread context are used.

9.8.2.5 int hpss_SetACL (const char ∗ *Path,* const uint32_t *Options,* const ns_ACLConfArray_t ∗ *ACL*)

Sets the ACL entries of a file or directory.

The 'hpss_SetACL' function sets the Access Control List entries of a file or directory named by 'Path'.

Parameters

in	*Path*	path of file
in	*Options*	bitmask of ACL options
in	*ACL*	ACL array to be set

Return values

0	No error.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path or ACL parameter is a NULL pointer.
-EINVAL	Exactly one of the HPSS_ACL_OPTION_∗ bits must One of the following has occurred: • More than one of the HPSS_ACL_OPTION_∗ bits are set. • ObjHandle is NULL. • HPSS_ACL_INITIAL_CONTAINER_ACL or HPSS_ACL_INITIAL_OBJECT_ACL was set and the ACL object is not a directory or a fileset root. be set in the Options bit vector.

-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation.

9.8.2.6 int hpss_SetACLHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* const uint32_t *Options,* const ns_ACLConfArray_t ∗ *ACL*)

Sets the ACL entries of a file or directory using a handle.

The ’hpss_SetACLHandle’ function sets the Access Control List entries of a file or directory named by ’Path’, taken relative to the directory indicated by ’ObjHandle’.

Parameters

`in`	*ObjHandle*	parent object handle
`in`	*Path*	path of file
`in`	*Ucred*	user credentials
`in`	*Options*	bitmask of ACL options
`in`	*ACL*	ACL array to be set

Return values

0	No error.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path or ACL parameter is a NULL pointer.
-EINVAL	Exactly one of the HPSS_ACL_OPTION_∗ bits must One of the following has occurred: • More than one of the HPSS_ACL_OPTION_∗ bits are set. • ObjHandle is NULL. • HPSS_ACL_INITIAL_CONTAINER_ACL or HPSS_ACL_INITIAL_OBJECT_ACL was set and the ACL object is not a directory or a fileset root. be set in the Options bit vector.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation.

9.8.2.7 int hpss_UpdateACL (const char ∗ *Path,* int32_t *Options,* const ns_ACLConfArray_t ∗ *ACL*)

Updates an ACL array of a file or directory.

The ’hpss_UpdateACL’ function updates an ACL array from the Access Control List of a file or directory named by ’Path’.

Parameters

in	*Path*	path of file
in	*Options*	processing options
in	*ACL*	ACL array to be updated

Return values

0	No error.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path or ACL parameter is a NULL pointer.
-EINVAL	There are two sets of mutually exclusive flags available through the Options paramter. An invalid combinations of flags were provided.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation.

9.8.2.8 int hpss_UpdateACLHandle (const ns_ObjHandle_t ∗ *ObjHandle*, const char ∗ *Path*, const sec_cred_t ∗ *Ucred*, uint32_t *Options*, const ns_ACLConfArray_t ∗ *ACL*)

Updates an ACL entry of a file or directory using a handle.

The 'hpss_UpdateACLHandle' function updates an ACL entry from the Access Control List of a file or directory named by 'Path', taken relative to the directory indicated by 'ObjHandle'.

Parameters

in	*ObjHandle*	parent object handle
in	*Path*	path of file
in	*Ucred*	user credentials
in	*Options*	processing options
in	*ACL*	ACL array to be updated

Return values

0	No error.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path or ACL parameter is a NULL pointer.
-EINVAL	There are two sets of mutually exclusive flags available through the Options paramter. An invalid combinations of flags were provided.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The client does not have the appropriate privileges to perform the operation.

9.9 Directory Creation and Deletion

Functions used to create, remove, and manipulate directories.

Functions

- int hpss_Mkdir (const char ∗Path, mode_t Mode)

  Creates a new directory.
- int hpss_MkdirHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, mode_t Mode, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗HandleOut, hpss_vattr_t ∗AttrOut)

  Creates a new directory.
- int hpss_Rmdir (const char ∗Path)

  Remove a directory.
- int hpss_RmdirHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Remove a directory using a handle.
- int hpss_RmdirHandleImmediate (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Remove a directory using a handle, bypassing the trashcan.
- int hpss_RmdirImmediate (const char ∗Path)

  Remove a directory, bypassing the trashcan.

9.9.1 Detailed Description

Functions used to create, remove, and manipulate directories.

9.9.2 Function Documentation

9.9.2.1 int hpss_Mkdir (const char ∗ *Path*, mode_t *Mode*)

Creates a new directory.

The 'hpss_Mkdir' function creates a new directory with the name 'Path'. The directory permission bits of the new directory are initialized by 'Mode', and modified by the file creation mask of the thread.

Parameters

`in`	*Path*	path of directory
`in`	*Mode*	permission bits of the new directory

Return values

0	No error. New directory created.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.
-EEXIST	The named file exists.
-EFAULT	The Path parameter is a NULL pointer.
-EMLINK	The link count of the parent directory would exceed the maximum allowed number of links.

-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the pathname exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

9.9.2.2 int hpss_MkdirHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* mode_t *Mode,* const sec_cred_t ∗ *Ucred,* ns_ObjHandle_t ∗ *HandleOut,* hpss_vattr_t ∗ *AttrOut*)

Creates a new directory.

The 'hpss_MkdirHandle' function creates a new directory with the name 'Path', taken relative to the directory indicated by 'ObjHandle'. The directory permission bits of the new directory are initialized by 'Mode', and modified by the file creation mask of the thread. The newly created directory's object handle and attributes are returned in the areas pointed to by 'RetObjHandle' and 'RetVAttrs', respectively.

Parameters

`in`	*ObjHandle*	handle of parent directory
`in`	*Path*	path of directory
`in`	*Mode*	perm bits for new directory
`in`	*Ucred*	user credentials
`out`	*HandleOut*	returned object handle
`out`	*AttrOut*	returned VFS attributes

Return values

0	No error. New directory created.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.
-EEXIST	The named file exists.
-EFAULT	The Path parameter is a NULL pointer.
-EMLINK	The link count of the parent directory would exceed the maximum allowed number of links.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the pathname exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

9.9.2.3 int hpss_Rmdir (const char ∗ *Path*)

Remove a directory.

The 'hpss_Rmdir' function removes the directory named by 'Path'. The directory will only be removed if the directory is empty.

Parameters

`in`	*Path*	path of directory

Return values

0	Success

-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.
-EBUSY	The named directory is currently in use and cannot be removed.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-ENOTEMPTY	The named directory contains entries other than dot and dot-dot.

Note

If trashcans are enabled, Path will overwritten by the rename, and be moved to the trashcan corresponding to to the deleting user and the Path's location.

9.9.2.4 int hpss_RmdirHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred*)

Remove a directory using a handle.

The 'hpss_RmdirHandle' function removes the directory named by 'Path', taken relative to the directory indicated by 'ObjHandle'. The directory will only be removed if the directory is empty.

Parameters

`in`	*ObjHandle*	parent object handle
`in`	*Path*	path of directory
`in`	*Ucred*	user credential

Return values

0	Success
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.
-EBUSY	The named directory is currently in use and cannot be removed.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	ObjHandle is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-ENOTEMPTY	The named directory contains entries other than dot and dot-dot.

Note

If trashcans are enabled, Path will overwritten by the rename, and be moved to the trashcan corresponding to to the deleting user and the Path's location.

9.9.2.5 int hpss_RmdirHandleImmediate (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred*)

Remove a directory using a handle, bypassing the trashcan.

The 'hpss_RmdirHandleImmediate' function removes the directory named by 'Path', Taken relative to the directory indicated by 'ObjHandle'. The directory will only be removed if the directory is empty.

Parameters

`in`	*ObjHandle*	parent object handle
`in`	*Path*	path of directory
`in`	*Ucred*	user credential

Return values

0	Success
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.
-EBUSY	The named directory is currently in use and cannot be removed.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	ObjHandle is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-ENOTEMPTY	The named directory contains entries other than dot and dot-dot.

Note

This directory will immediately be deleted, regardless of trashcan settings.

9.9.2.6 int hpss_RmdirImmediate (const char ∗ *Path*)

Remove a directory, bypassing the trashcan.

The 'hpss_RmdirImmediate' function removes the directory named by 'Path'. The directory will only be removed if the directory is empty.

Parameters

`in`	*Path*	path of directory

Return values

0	Success
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.
-EBUSY	The named directory is currently in use and cannot be removed.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-ENOTEMPTY	The named directory contains entries other than dot and dot-dot.

Note

This directory will immediately be deleted, regardless of trashcan settings.

9.10 Directory Data

Functions used to access directory listings.

Functions

- int hpss_Closedir (int Dirdes)

  Close an open directory stream.
- int hpss_Getdents (const ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_t ∗Ucred, uint32_t BufferSize, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

  Retrieve directory entries with reliable entry offsets.
- int hpss_GetdentsAttrs (ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_t ∗Ucred, uint32_t BufferSize, uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entries and object attributes in an offset-reliable manner.
- int hpss_Opendir (const char ∗DirName)

  Opens a directory stream.
- int hpss_OpendirHandle (const ns_ObjHandle_t ∗DirHandle, const sec_cred_t ∗Ucred)

  Open a directory via a handle.
- int hpss_ReadAttrs (const int Dirdes, const uint64_t OffsetIn, const uint32_t BufferSize, const uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entries and object attributes from an open directory stream.
- int hpss_ReadAttrsHandle (const ns_ObjHandle_t ∗ObjHandle, const uint64_t OffsetIn, const sec_cred_t ∗Ucred, const uint32_t BufferSize, const uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entry attributes using a handle.
- int hpss_ReadAttrsPlus (int Dirdes, uint64_t OffsetIn, uint32_t BufferSize, hpss_readdir_flags_t Flags, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entries and object attributes from an open directory stream.
- int hpss_Readdir (int Dirdes, hpss_dirent_t ∗DirentPtr)

  Read directory entries from an open directory stream.
- int hpss_ReaddirHandle (const ns_ObjHandle_t ∗ObjHandle, const uint64_t OffsetIn, const sec_cred_t ∗Ucred, const uint32_t BufferSize, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

  Read directory entries using a handle.
- int hpss_ReaddirPlus (int Dirdes, uint64_t OffsetIn, uint32_t BufferSize, hpss_readdir_flags_t Flags, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

  Read directory entries using a handle.
- int hpss_ReadRawAttrsHandle (const ns_ObjHandle_t ∗ObjHandle, const uint64_t OffsetIn, const sec_cred_t ∗Ucred, const uint32_t BufferSize, const uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read raw attributes using a handle; does not cross junctions.
- int hpss_Rewinddir (int Dirdes)

  Rewind a directory stream back to the beginning.

9.10.1 Detailed Description

Functions used to access directory listings.

9.10.2 Function Documentation

9.10.2.1 int hpss_Closedir (int *Dirdes*)

Close an open directory stream.

Parameters

`in`	*Dirdes*	Open directory stream handle

The 'hpss_CloseDir' function closes the directory stream corresponding to the open directory stream handle 'Dirdes'.

Return values

0	No error.
-EBADF	The specified directory descriptor does not refer to an open directory.
-EBUSY	The directory is currently in use by another client thread.

See Also

hpss_OpenDir

9.10.2.2 int hpss_Getdents (const **ns_ObjHandle_t** ∗ *ObjHandle,* uint64_t *OffsetIn,* const sec_cred_t ∗ *Ucred,* uint32_t *BufferSize,* uint32_t ∗ *End,* uint64_t ∗ *OffsetOut,* **hpss_dirent_t** ∗ *DirentPtr*)

Retrieve directory entries with reliable entry offsets.

The 'hpss_Getdents' function fills in the passed buffer with directory entries beginning at the specified directory position. This call differs from the other readdir calls in that it will return reliable entry offsets that can be used to resume querying the directory across closes.

Parameters

`in`	*ObjHandle*	directory object handle
`in`	*OffsetIn*	directory position
`in`	*Ucred*	user credentials
`in`	*BufferSize*	size of output buffer
`out`	*End*	hit end of directory
`out`	*OffsetOut*	resulting directory position
`out`	*DirentPtr*	directory entry information

Return values

0	Success
-EBADF	The specified directory descriptor does not refer to an open directory.
-EBUSY	The directory is currently in use by another client thread.
-EFAULT	The DirentPtr, End or OffsetOut parameter is a NULL pointer.
-EINVAL	The ObjHandle pointer is NULL or the BufferSize parameter is zero.

Deprecated This function will be removed in a future version of HPSS. Use hpss_ReaddirPlus().

9.10.2.3 int hpss_GetdentsAttrs (**ns_ObjHandle_t** ∗ *ObjHandle,* uint64_t *OffsetIn,* const sec_cred_t ∗ *Ucred,* uint32_t *BufferSize,* uint32_t *GetAttributes,* uint32_t ∗ *End,* uint64_t ∗ *OffsetOut,* **ns_DirEntry_t** ∗ *DirentPtr*)

Read directory entries and object attributes in an offset-reliable manner.

The 'hpss_GetdentsAttrs' function fills in the passed buffer with directory entries, include file/directory attributes, beginning at the specified directory position. This call differs from the other readdir calls in that it will return reliable entry offsets that can be used to resume querying the directory across closes.

Parameters

in	*ObjHandle*	directory object handle
in	*OffsetIn*	directory position
in	*Ucred*	user credentials
in	*BufferSize*	size of output buffer
in	*GetAttributes*	get object attributes?
out	*End*	hit end of directory
out	*OffsetOut*	resulting directory position
out	*DirentPtr*	directory entry information

Returns

Diretory entry. If null string is returned in d_name and/or d_namelen ==0, we hit end of directory

Return values

0	Success
-EBADF	The specified directory descriptor does not refer to an open directory.
-EBUSY	The directory is currently in use by another client thread.
-EFAULT	The DirentPtr, End or OffsetOut parameter is a NULL pointer.
-EINVAL	The ObjHandle pointer is NULL or the BufferSize parameter is zero.

Deprecated This function will be removed in a future version of HPSS. Use hpss_ReadAttrsPlus().

9.10.2.4 int hpss_Opendir (const char ∗ *DirName*)

Opens a directory stream.

The 'hpss_Opendir' function opens a directory stream corresponding to the directory named by 'DirName'. The directory stream is positioned at the first entry in the directory.

Parameters

in	*DirName*	path of directory

Return values

0	Success
-EACCES	Search permission is denied on a component of the path prefix, or read permission is denied on the directory itself.
-EFAULT	The DirName parameter is a NULL pointer.
-EMFILE	The open file table is already full.
-ENAMETOOLONG	The length of the DirName argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the DirName argument points to an empty string.
-ENOTDIR	A component of DirName is not a directory.

9.10.2.5 int hpss_OpendirHandle (const ns_ObjHandle_t ∗ *DirHandle,* const sec_cred_t ∗ *Ucred*)

Open a directory via a handle.

Parameters

in	*DirHandle*	Directory Handle
in	*Ucred*	User credential

Return values

-EFAULT	No handle provided
>=0	Open directory descriptor

Note

If a NULL *Ucred* is provided, the thread credentials will be used.

9.10.2.6 int hpss_ReadAttrs (const int *Dirdes,* const uint64_t *OffsetIn,* const uint32_t *BufferSize,* const uint32_t *GetAttributes,* uint32_t ∗ *End,* uint64_t ∗ *OffsetOut,* ns_DirEntry_t ∗ *DirentPtr*)

Read directory entries and object attributes from an open directory stream.

The 'hpss_ReadAttrs' function fills in the passed buffer with directory entries, including file/directory attributes, beginning at the specified directory position.

Parameters

in	*Dirdes*	open directory stream handle
in	*OffsetIn*	directory position
in	*BufferSize*	size of output buffer
in	*GetAttributes*	get object attributes
out	*End*	hit end of directory
out	*OffsetOut*	resulting directory position
out	*DirentPtr*	directory entry information

Return values

0	No error. 'DirentPtr' contains directory information.
-EBADF	The specified directory descriptor does not refer to an open directory.
-EBUSY	The directory is currently in use by another client thread.
-EFAULT	The DirentPtr, End or OffsetOut parameter is a NULL pointer.
-EINVAL	The BufferSize parameter is zero.

Deprecated This function will be removed in a future version of HPSS. Use hpss_ReadAttrsPlus().

9.10.2.7 int hpss_ReadAttrsHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const uint64_t *OffsetIn,* const sec_cred_t ∗ *Ucred,* const uint32_t *BufferSize,* const uint32_t *GetAttributes,* uint32_t ∗ *End,* uint64_t ∗ *OffsetOut,* ns_DirEntry_t ∗ *DirentPtr*)

Read directory entry attributes using a handle.

The 'hpss_ReadAttrsHandle' function fills in the passed buffer with directory entries, include file/directory attributes, beginning at the specified directory position.

Parameters

`in`	*ObjHandle*	directory object handle
`in`	*OffsetIn*	directory position
`in`	*Ucred*	user credentials
`in`	*BufferSize*	size of output buffer
`in`	*GetAttributes*	get object attributes
`out`	*End*	hit end of directory
`out`	*OffsetOut*	resulting directory position
`out`	*DirentPtr*	directory entry information

Returns

DirentPtr: Directory entry. If null string is returned in d_name and/or d_namelen == 0, we hit end of directory.

Return values

0	No error. 'DirentPtr' contains directory information
-EBADF	The specified directory descriptor does not refer to an open directory.
-EBUSY	The directory is currently in use by another client thread.
-EFAULT	The DirentPtr, End or OffsetOut parameter is a NULL pointer.
-EINVAL	The ObjHandle pointer is NULL or the BufferSize parameter is zero.

Deprecated This function will be removed in a future version of HPSS. Use hpss_ReadAttrsPlus().

9.10.2.8 int hpss_ReadAttrsPlus (int *Dirdes,* uint64_t *OffsetIn,* uint32_t *BufferSize,* hpss_readdir_flags_t *Flags,* uint32_t ∗ *End,* uint64_t ∗ *OffsetOut,* ns_DirEntry_t ∗ *DirentPtr*)

Read directory entries and object attributes from an open directory stream.

The 'hpss_ReadAttrsPlus' function fills in the passed buffer with directory entries, including file/directory attributes, beginning at the specified directory position.

Parameters

`in`	*Dirdes*	open directory stream handle
`in`	*OffsetIn*	directory position
`in`	*BufferSize*	size of output buffer
`in`	*Flags*	readdir flags
`out`	*End*	hit end of directory
`out`	*OffsetOut*	resulting directory position
`out`	*DirentPtr*	directory entry information

Return values

$>$*0*	Numberof directory entries returned.
0	No error. 'DirentPtr' contains directory information.
-EBADF	The specified directory descriptor does not refer to an open directory.
-EBUSY	The directory is currently in use by another client thread.
-EFAULT	The DirentPtr, End or OffsetOut parameter is a NULL pointer.
-EINVAL	The BufferSize parameter is zero.

9.10.2.9 int hpss_Readdir (int *Dirdes,* hpss_dirent_t ∗ *DirentPtr*)

Read directory entries from an open directory stream.

The 'hpss_Readdir' function returns a structure representing the directory entry at the current position in the open directory stream.

Parameters

`in`	*Dirdes*	open directory stream handle
`out`	*DirentPtr*	directory entry information

Returns

DirentPtr: Directory entry. If null string is returned in d_name and/or d_namelen == 0, we hit end of directory.

Return values

0	No error. 'DirentPtr' points to the entry read
-EBADF	The specified directory descriptor does not refer to an open directory.
-EBUSY	The directory is currently in use by another client thread.
-EFAULT	The DirentPtr parameter is a NULL pointer.

Note

This is function is generally inefficient for operations which follow a readdir -> stat -> readdir -> stat pattern for getting dirent attributes followed by file attributes. Functions such as hpss_ReadAttrs() and hpss_ReadAttrPlus() are provided for more efficient means executing this pattern.

9.10.2.10 int hpss_ReaddirHandle (const **ns_ObjHandle_t** ∗ *ObjHandle,* const uint64_t *OffsetIn,* const sec_cred_t ∗ *Ucred,* const uint32_t *BufferSize,* uint32_t ∗ *End,* uint64_t ∗ *OffsetOut,* **hpss_dirent_t** ∗ *DirentPtr*)

Read directory entries using a handle.

The 'hpss_ReaddirHandle' function fills in the passed buffer with directory entries beginning at the specified directory position.

Parameters

`in`	*ObjHandle*	directory object handle
`in`	*OffsetIn*	directory position
`in`	*Ucred*	user credentials
`in`	*BufferSize*	size of output buffer
`out`	*End*	hit end of directory
`out`	*OffsetOut*	resulting directory position
`out`	*DirentPtr*	directory entry information

Returns

Directory entry. If null string is returned in d_name and/or d_namelen == 0, we hit end of directory.

Return values

>=0	Number of entries read from the Core Server
-EBADF	The specified directory descriptor does not refer to an open directory.
-EBUSY	The directory is currently in use by another client thread.

-EFAULT	One of the End, OffsetOut, or DirentPtr parameters is NULL.
-EINVAL	BufferSize is zero or ObjHandle is NULL.

Deprecated This function will be removed in a future version of HPSS. Use hpss_ReaddirPlus().

9.10.2.11 int hpss_ReaddirPlus (int *Dirdes,* uint64_t *OffsetIn,* uint32_t *BufferSize,* hpss_readdir_flags_t *Flags,* uint32_t ∗ *End,* uint64_t ∗ *OffsetOut,* hpss_dirent_t ∗ *DirentPtr*)

Read directory entries using a handle.

The 'hpss_ReaddirPlus' function fills in the passed buffer with directory entries beginning at the specified directory position.

Parameters

`in`	*Dirdes*	Open directory descriptor
`in`	*OffsetIn*	directory position
`in`	*BufferSize*	size of output buffer
`in`	*Flags*	readdir flags
`out`	*End*	hit end of directory
`out`	*OffsetOut*	resulting directory position
`out`	*DirentPtr*	directory entry information

Returns

Directory entry. If null string is returned in d_name and/or d_namelen == 0, we hit end of directory.

Return values

>0	Number of entries read from the Core Server
==0	Hit end of directory
-EBADF	The specified directory descriptor does not refer to an open directory.
-EBUSY	The directory is currently in use by another client thread.
-EFAULT	One of the End, OffsetOut, or DirentPtr parameters is NULL.
-EINVAL	BufferSize is zero or ObjHandle is NULL.

9.10.2.12 int hpss_ReadRawAttrsHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const uint64_t *OffsetIn,* const sec_cred_t ∗ *Ucred,* const uint32_t *BufferSize,* const uint32_t *GetAttributes,* uint32_t ∗ *End,* uint64_t ∗ *OffsetOut,* ns_DirEntry_t ∗ *DirentPtr*)

Read raw attributes using a handle; does not cross junctions.

The 'hpss_ReadRawAttrsHandle' function fills in the passed buffer with directory entries, include file/directory attributes, beginning at the specified directory position. This routine will NOT try to chase HPSS junctions encountered.

Parameters

`in`	*ObjHandle*	directory object handle
`in`	*OffsetIn*	directory position
`in`	*Ucred*	user credentials
`in`	*BufferSize*	size of output buffer
`in`	*GetAttributes*	get object attributes
`out`	*End*	hit end of directory
`out`	*OffsetOut*	resulting directory position
`out`	*DirentPtr*	directory entry information

Returns

DirentPtr: Directory entry. If null string is returned in d_name and/or d_namelen == 0, we hit end of directory.

Return values

0	No error. 'DirentPtr' contains directory information
-EBADF	The specified directory descriptor does not refer to an open directory.
-EBUSY	The directory is currently in use by another client thread.
-EFAULT	The DirentPtr, End or OffsetOut parameter is a NULL pointer.
-EINVAL	The ObjHandle pointer is NULL or the BufferSize parameter is zero.

Deprecated This function will be removed in a future version of HPSS. Use hpss_ReadAttrsPlus().

9.10.2.13 int hpss_Rewinddir (int *Dirdes*)

Rewind a directory stream back to the beginning.

Parameters

`in`	*Dirdes*	open directory stream handle

The 'hpss_Rewinddir' function resets the position of an open directory stream corresponding to 'Dirdes' to the beginning of that directory.

Return values

0	Success
-EBADF	The specified directory descriptor does not correspond to an open directory or Dirdes is negative, or there are too many open file and directory descriptors.
-EBUSY	Another client thread is currently using this directory descriptor.
-ESTALE	Connection for this entry no longer valid.

9.11 Working Directory

Functions used to determine or modify the working directory.

Functions

- int hpss_Chdir (const char ∗Path)

  Changes the current working directory.
- int hpss_Chroot (const char ∗Path)

  Sets the client's root directory.
- int hpss_Getcwd (char ∗Buf, size_t Size)

  Retrieves the current working directory.
- int hpss_GetThreadUcred (sec_cred_t ∗RetUcred)

  Retrieves the user credentials for the current thread.

9.11.1 Detailed Description

Functions used to determine or modify the working directory.

9.11.2 Function Documentation

9.11.2.1 int hpss_Chdir (const char ∗ *Path*)

Changes the current working directory.

The 'hpss_Chdir' function changes the thread's current working directory to be the directory named by *Path*.

Parameters

`in`	*Path*	Path to the directory

Return values

0	No error. New working directory set properly.
-EACCES	Search permission is denied on a component of the *Path* prefix.
-EFAULT	The *Path* parameter is a NULL pointer.
-ENAMETOOLONG	The length of the *Path* argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the *Path* argument points to an empty string.
-ENOTDIR	A component of the *Path* prefix is not a directory.

9.11.2.2 int hpss_Chroot (const char ∗ *Path*)

Sets the client's root directory.

The 'hpss_Chroot' function sets the client's root directory. From this point, all directory operations (following a cd into this directory, if necessary) will be bounded by this new root directory.

Parameters

in	*Path*	path to the object

Return values

0	No error. Changes made correctly.
-EACCES	Search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	This call was made from the non-global Client API library.
-EINVAL	Could not normalize the path
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-ENOMEM	Memory allocation failure

Note

As currently implemented, symbolic links allow a client to access files outside the new root directory.

9.11.2.3 int hpss_Getcwd (char ∗ *Buf,* size_t *Size*)

Retrieves the current working directory.

The 'hpss_Getcwd' function copies an absolute pathname of the current working directory to the character array pointed to by 'Buf'. The 'Size' argument is the size in bytes of the array pointed to by 'Buf'.

Parameters

out	*Buf*	current working directory
in	*Size*	size of the buffer

Return values

0	No error.
-EACCES	Read or Search permission is denied on a component of the path name.
-EFAULT	The Buf parameter is a NULL pointer.
-EINVAL	The Size argument is zero.
-ERANGE	The Size argument is greater than zero, but smaller than the length of the path name plus 1.

9.11.2.4 int hpss_GetThreadUcred (sec_cred_t ∗ *RetUcred*)

Retrieves the user credentials for the current thread.

The 'hpss_GetThreadUcred' function is used to get the user credentials for the currently running thread.

Parameters

out	*RetUcred*	credentials on client

Return values

0	No error. The RetUcred contains the current credentials for the user.
-EFAULT	RetUcred is a NULL pointer.

9.12 Client API Control

Functions used to control Client API state.

Functions

- int API_GetDataThreadStackSize (void)

  Retrieves the data thread stack size.
- void API_SocketSetOptions (int Socket, hpss_sockaddr_t ∗SaddrPtr, unsigned int Options)

  Sets the API socket options.
- int hpss_ClientAPIInit (void)

  Initialize the Client API for the current thread.
- void hpss_ClientAPIReset ()

  Resets the client API state.
- int hpss_GetConfiguration (api_config_t ∗ConfigOut)

  Retrieves the current values used for default configurations.
- int hpss_LoadDefaultThreadState (uid_t UserID, mode_t Umask, const char ∗ClientFullName)

  Allows some clients to manipulate the global state of a thread.
- int hpss_MiscAdmin (uint32_t Function, char ∗TextP, ns_AdminConfArray_t ∗ValueConfArrayP)

  Accesses the Core Server's miscellaneous functions.
- int hpss_SetAPILogLevel (int LogLevel)

  Modifies the Client API log level.
- int hpss_SetAPILogPath (const char ∗LogPath)

  Modifies the Client API log path.
- void hpss_SetApplicationName (const char ∗App)

  Modifies the Client API application name.
- int hpss_SetConfiguration (const api_config_t ∗ConfigIn)

  Sets the values used for the API configuration settings.
- int hpss_ThreadCleanUp ()

  Cleans up a thread's context.

9.12.1 Detailed Description

Functions used to control Client API state.

9.12.2 Function Documentation

9.12.2.1 int API_GetDataThreadStackSize (void)

Retrieves the data thread stack size.

This routine returns the chosen data thread stack size.

Returns

The default data thread stack size

See Also

hpss_GetConfiguration

9.12.2.2 void API_SocketSetOptions (int *Socket,* hpss_sockaddr_t ∗ *SaddrPtr,* unsigned int *Options*)

Sets the API socket options.

Parameters

`in`	*Socket*	Socket descriptor
`in`	*SaddrPtr*	Socket address
`in`	*Options*	Socket options

The API_SocketSetOptions routine is called to set the send and receive buffer sizes, TCP no delay and keepalive flags.

Return values

None.	

9.12.2.3 int hpss_ClientAPIInit (void)

Initialize the Client API for the current thread.

This is usually done automatically during the first Client API call made. However it may also be performed separately, before other Client API calls. This function does nothing if the Client API is already initialized.

Return values

0	Success
-ENOCONNECT	Could not initialize HPSS connection services.
-ENOMEM	Memory allocation failed.
-EPERM	Could not initialize security.

9.12.2.4 void hpss_ClientAPIReset (void)

Resets the client API state.

This routine will reset the client API state, including closing all server connections and cleaning up based on environment variables. This will allow a subsequent client API call to re-initialize the API based on current login context and environment variables.

See Also

hpss_GetConfiguration hpss_SetConfiguration

9.12.2.5 int hpss_GetConfiguration (api_config_t * *ConfigOut*)

Retrieves the current values used for default configurations.

This routine will return the current settings of the default configuration parameters.

Parameters

`out`	*ConfigOut*	ptr for returned attributes

Return values

0	Success
-EFAULT	The ConfigOut parameter is NULL.

9.12.2.6 int hpss_LoadDefaultThreadState (uid_t *UserID,* mode_t *Umask,* const char ∗ *ClientFullName*)

Allows some clients to manipulate the global state of a thread.

Special interface to allow well-behaved clients to manipulate the global thread state so that all threads will pickup the Ucreds and Umask from the Global Thread State. After this call, hpss_LoadThreadState() is effectively disabled because for each API the credentials (and mask) are loaded from the Global Thread State. If a non-NULL 'ClientFullName' is passed in, then the user's credentials are obtained using the clients ID. This call can be used by clients that have multiple threads with the same identity.

Parameters

in	*UserID*	New user credentials
in	*Umask*	New umask
in	*ClientFullName*	Client fully qualified name

Return values

-EAGAIN	Default security registry error.
-EINVAL	NumGroups is invalid
-ENOCONNECT	Security registry is unavailable.
-ENOENT	No thread with given ThreadID or security registry entry was not found.
-ENOMEM	Unable to allocate memory
0	No error. Thread specific state is valid.

Note

The effects of this call aren't reversed by resetting the API.

9.12.2.7 int hpss_MiscAdmin (uint32_t *Function,* char ∗ *TextP,* ns_AdminConfArray_t ∗ *ValueConfArrayP*)

Accesses the Core Server's miscellaneous functions.

Parameters

in	*Function*	Func to perform.
in,out	*TextP*	For auxil text.
in,out	*ValueConfArray-P*	For auxil data.

The hpss_MiscAdmin function allows access to the Core Server's miscellaneous functions. This API is used by "Trusted Clients" to perform various administrative tasks within the Core Server. These tasks are defined by the value of the Function parameter and, in addition, the Function may be modified or further defined by the values found in TextP and/or ValueConfArrayP. In other words, if any of the functions require more input, TextP and/or ValueConfArrayP are used. And, in addition, if any of the functions require output, TextP and/or ValueConfArrayP are used.

The available Function values are defined by the following constants–

- NS_ADMIN_CLEAR_ALL_ADMIN_FLAGS - clear bit vector used by NS.
- NS_ADMIN_DUMP_CORE_SERVER_STATE - send all CS state to a file.
- NS_ADMIN_DUMP_DISK_MAPS - send the CS diskmaps to a file.
- NS_ADMIN_FLIP_BFS_EMPTY_COS - flip BFS's boolean empty COS bit.
- NS_ADMIN_HALT_THE_CORE_SERVER _ immediately halt the CS.
- NS_ADMIN_PERF_LOGGING_OFF - performance logging is turned off.

- NS_ADMIN_PERF_LOGGING_ON - performance looging is turned on.
- NS_ADMIN_PERF_LOGGING_STAT - get status of performance logging.
- NS_ADMIN_PRINT_LOG_MSGS - print selected log messages.
- NS_ADMIN_REINIT_FS_CACHE - reinitialize the fileset cache.
- NS_ADMIN_SET_COMMUNICATION_STATE- set the CS communication state.
- NS_ADMIN_SET_OPERATIONAL_STATE - set the CS operational state.
- NS_ADMIN_SET_SOFTWARE_STATE - set the CS software state.

Note

The TextP is used to input or output any text that might be required by a particular Function. The Value-ConfArray is a conformant array of uint64_t's and is used to pass any numerical data required or output by a Function.

Return values

0	No error. User List in buffer.
-EINVAL	The SiteName parameter is invalid
-EFAULT	The ServerId parameter is NULL
-ENOENT	The Root Core Server could not be located

9.12.2.8 int hpss_SetAPILogLevel (int *LogLevel*)

Modifies the Client API log level.

Parameters

in	*LogLevel*	Client API Log Level • API_DEBUG_NONE for no logging • API_DEBUG_ERROR for error logging • API_DEBUG_REQUEST for request logging • API_DEBUG_TRACE for trace logging

Return values

0	Success

9.12.2.9 int hpss_SetAPILogPath (const char ∗ *LogPath*)

Modifies the Client API log path.

Parameters

in	*LogPath*	Client API Log Path

Return values

0	Success
EACCES	Requested access is not allowed
EFAULT	LogPath is outside addressable address space
EISDIR	LogPath refers to a directory
ELOOP	Too many symbolic links were encountered
EMFILE	The process already has the maximum number of files open.
ENAMETOOLONG	LogPath is too long.
ENOSPC	No space for the new logfile
ENOTDIR	A component used as a directory in LogPath was not a directory
EOVERFLOW	LogPath refers to a regular file that is too large to be opened
EROFS	LogPath refers to a file on a read-only file system
ENOMEM	Ran out of memory

9.12.2.10 void hpss_SetApplicationName (const char ∗ *App*)

Modifies the Client API application name.

Parameters

`in`	*App*	Application name string

9.12.2.11 int hpss_SetConfiguration (const api_config_t ∗ *ConfigIn*)

Sets the values used for the API configuration settings.

This routine will set the current settings of the default configuration parameters.

Parameters

`in`	*ConfigIn*	Configuration parameters

Return values

0	Success
-EFAULT	The ConfigIn parameter is NULL
-EINVAL	Invalid configuration attribute value setting

9.12.2.12 int hpss_ThreadCleanUp (void)

Cleans up a thread's context.

This routine is called to clean up a thread's context.

Return values

0	Success

9.13 Authentication

Functions used to authenticate the Client API with HPSS.

Functions

- void hpss_PurgeLoginCred (void)

  Purges an HPSS login credential.
- int hpss_SetAppLoginCred (const char ∗AppName, const char ∗PrincipalName, hpss_authn_mech_t Mechanism, hpss_rpc_cred_type_t CredType, hpss_rpc_auth_type_t AuthType, const void ∗Authenticator)

  Sets an HPSS login credential.
- int hpss_SetAuditInfo (hpss_sockaddr_t EndUserHostAddr)

  Sets audit information for threads.
- int hpss_SetLoginCred (const char ∗PrincipalName, hpss_authn_mech_t Mechanism, hpss_rpc_cred_type_t CredType, hpss_rpc_auth_type_t AuthType, const void ∗Authenticator)

  Sets an HPSS login credential.

9.13.1 Detailed Description

Functions used to authenticate the Client API with HPSS.

9.13.2 Function Documentation

9.13.2.1 void hpss_PurgeLoginCred (void)

Purges an HPSS login credential.

This routine is called to purge an HPSS login credential. The purge call must be issued to end update of the specified user credential.

Return values

None.	

Note

This function is not thread safe, and they should only be called within a single thread, or serialized using a mutex.

9.13.2.2 int hpss_SetAppLoginCred (const char ∗ *AppName,* const char ∗ *PrincipalName,* hpss_authn_mech_t *Mechanism,* hpss_rpc_cred_type_t *CredType,* hpss_rpc_auth_type_t *AuthType,* const void ∗ *Authenticator*)

Sets an HPSS login credential.

This routine is called to set an HPSS login credential.

Parameters

in	*AppName*	Application name
in	*PrincipalName*	Principal name
in	*Mechanism*	Security Mechanism
in	*CredType*	Type of creds needed
in	*AuthType*	Authenticator type
in	*Authenticator*	Authenticator

Return values

0	Success
Otherwise	An other HPSS errno indicatingnature of the error

Note

A newly set credential will be refreshed in the background and the purge call must be issued to end update of the specified user credientials. This function is not thread safe, and they should only be called within a single thread, or serialized using a mutex.

9.13.2.3 int hpss_SetAuditInfo (hpss_sockaddr_t *EndUserHostAddr*)

Sets audit information for threads.

This routine sets up per-thread information that will appear in Core Server security audit messages.

Parameters

in	*EndUserHost-Addr*	enduser's host IP address

Note

This routine should be called by every thread that wants to use per-thread audit information. If the routine is never called, then the audit messages will show the host address of the server (eg., ftpd) that initiated the request. This is typically not very helpful; so the servers should call the routine at least once. If the routine is called once by one thread but not by another thread, then the audit messages will show the host address established by the first thread. This features lets ftpd call the routine once in the main thread rather than forcing ftpd to call it in every thread that contacts Core Server.

Return values

0	success
Non-zero	An error is returned from API_ClientAPIInit

9.13.2.4 int hpss_SetLoginCred (const char ∗ *PrincipalName*, hpss_authn_mech_t *Mechanism*, hpss_rpc_cred_type_t *CredType*, hpss_rpc_auth_type_t *AuthType*, const void ∗ *Authenticator*)

Sets an HPSS login credential.

This routine is called to set an HPSS login credential.

Deprecated This function is deprecated in 10.1. Use hpss_SetAppLoginCred instead.

Parameters

in	*PrincipalName*	Principal name
in	*Mechanism*	Security Mechanism
in	*CredType*	Type of creds needed
in	*AuthType*	Authenticator type
in	*Authenticator*	Authenticator

Return values

0	Success
Otherwise	An other HPSS errno indicatingnature of the error

Note

A newly set credential will be refreshed in the background and the purge call must be issued to end update of the specified user crediential. This function is not thread safe, and they should only be called within a single thread, or serialized using a mutex.

9.14 HPSS Statistics

Functions used to retrieve or reset HPSS statistics.

Functions

- int hpss_GetSubSysLimits (uint32_t SubsystemID, const sec_cred_t ∗Ucred, hpss_subsys_limits_t ∗LimitsOut)

  Retrieve subsystem limits.
- int hpss_GetSubSysStats (uint32_t SubsystemID, subsys_stats_t ∗StatsOut)

  Retrieve subsystem statistics.
- int hpss_ResetSubSysStats (uint32_t SubsystemID, subsys_stats_t ∗StatsOut)

  Reset subsystem statistics.
- int hpss_Statfs (uint32_t CosId, hpss_statfs_t ∗StatfsBuffer)

  Retrieve file system status information for a class of service.
- int hpss_Statfs64 (uint32_t CosId, hpss_statfs64_t ∗StatfsBuffer)

  Retrieve file system status information for a class of service.
- int hpss_Statvfs (uint32_t CosId, hpss_statvfs_t ∗StatvfsBuffer)

  Retrieve file system status information for a class of service in VFS format.
- int hpss_Statvfs64 (uint32_t CosId, hpss_statvfs64_t ∗StatvfsBuffer)

  Retrieve file system status information for a class of service in VFS format.

9.14.1 Detailed Description

Functions used to retrieve or reset HPSS statistics.

9.14.2 Function Documentation

9.14.2.1 int hpss_GetSubSysLimits (uint32_t *SubsystemID*, const sec_cred_t ∗ *Ucred*, hpss_subsys_limits_t ∗ *LimitsOut*)

Retrieve subsystem limits.

The 'hpss_GetSubSysLimits' function queries the Core Server for the current limits, status, and options.

Parameters

in	*SubsystemID*	Identifier for the subsystem
in	*Ucred*	User Credentials
out	*LimitsOut*	Subsystem limits information

Return values

0	No error.
-EFAULT	The LimitsOut parameter is a NULL pointer.

Note

Any of the data reported by this interface is subject to change at any time.

9.14.2.2 int hpss_GetSubSysStats (uint32_t *SubsystemID*, subsys_stats_t ∗ *StatsOut*)

Retrieve subsystem statistics.

The 'hpss_GetSubSysStats' function queries the Core Server for the stage, migration, purge and delete counts and the time these counts were last reset.

Parameters

`in`	*SubsystemID*	identifier for the subsystem
`out`	*StatsOut*	subsystem statistics information

Return values

0	No error.
-EFAULT	The StatsOut parameter is a NULL pointer.

9.14.2.3 int hpss_ResetSubSysStats (uint32_t *SubsystemID*, subsys_stats_t ∗ *StatsOut*)

Reset subsystem statistics.

The 'hpss_ResetSubSysStats' function resets the stage, migration, purge and delete counts in the Core Server and sets the last reset time to the current time.

Parameters

`in`	*SubsystemID*	identifier for the subsystem
`out`	*StatsOut*	subsystem statistics information

Return values

0	No error.
-EFAULT	The StatsOut parameter is a NULL pointer.

9.14.2.4 int hpss_Statfs (uint32_t *CosId*, hpss_statfs_t ∗ *StatfsBuffer*)

Retrieve file system status information for a class of service.

The 'hpss_Statfs' returns file system status information, in the structure pointed to by 'StatfsBuffer'. It retrieves this information from the location server in a list of all the statistics for this Class of Service from each Core Server which manages files in the Class of Service, and adds up the totals. The block size is that of the top storage class in the hierarchy.

Parameters

`in`	*CosId*	Class of service id.
`out`	*StatfsBuffer*	file system status.

Return values

0	Success
-EFAULT	The StatfsBuffer parameter is a NULL pointer.
-EINVAL	The specified Class of Service does not exist.

Deprecated This function will be removed in a future version of HPSS. Use hpss_Statvfs64().

9.14.2.5 int hpss_Statfs64 (uint32_t *CosId,* hpss_statfs64_t ∗ *StatfsBuffer*)

Retrieve file system status information for a class of service.

The 'hpss_Statfs64' returns 64-bit file system status information, in the structure pointed to by 'StatfsBuffer'. It retrieves this information from the location server in a list of all the statistics for this Class of Service from each Core Server which manages files in the Class of Service, and adds up the totals. The block size is that of the top storage class in the hierarchy.

Parameters

in	*CosId*	Class of service id.
out	*StatfsBuffer*	file system status.

Return values

0	Success
-EFAULT	The StatfsBuffer parameter is a NULL pointer.
-EINVAL	The specified Class of Service does not exist.

9.14.2.6 int hpss_Statvfs (uint32_t *CosId,* hpss_statvfs_t ∗ *StatvfsBuffer*)

Retrieve file system status information for a class of service in VFS format.

The 'hpss_Statvfs' returns file system status information, in the structure pointed to by 'StatvfsBuffer'. It retrieves this information from the location server in a list of all the statistics for this Class of Service from each Core Server which manages files in the Class of Service, and adds up the totals. The block size is that of the top storage class in the hierarchy.

Parameters

in	*CosId*	Class of service id.
out	*StatvfsBuffer*	file system status.

Return values

0	No error.
-EFAULT	The StatvfsBuffer parameter is a NULL pointer.
-EINVAL	The specified Class of Service does not exist.

Deprecated This function will be removed in a future version of HPSS. Use hpss_Statvfs64.

9.14.2.7 int hpss_Statvfs64 (uint32_t *CosId,* hpss_statvfs64_t ∗ *StatvfsBuffer*)

Retrieve file system status information for a class of service in VFS format.

The 'hpss_Statvfs64' returns 64-bit file system status information, in the structure pointed to by 'StatvfsBuffer'. It retrieves this information from the location server in a list of all the statistics for this Class of Service from each Core Server which manages files in the Class of Service, and adds up the totals. The block size is that of the top storage class in the hierarchy.

Parameters

`in`	*CosId*	Class of service id.
`out`	*StatvfsBuffer*	file system status.

Return values

0	No error.
-EFAULT	The StatvfsBuffer parameter is a NULL pointer.
-EINVAL	The specified Class of Service does not exist.

9.15 HPSS Object

Functions used to manipulate or construct HPSS objects.

Functions

- int hpss_GetObjFromHandle (const ns_ObjHandle_t ∗ObjHandle, object_id_hash_t ∗ObjHash)

    Retrieves an object from a handle.
- uint32_t hpss_GetObjType (const ns_ObjHandle_t ∗ObjHandle)

    Retrieves the object type given its handle.
- int hpss_GetPathHandle (const ns_ObjHandle_t ∗ObjHandle, ns_ObjHandle_t ∗FilesetRootHandle, char ∗Path)

    Retrieves the pathname and root fileset that correspond with an Objhandle.
- int hpss_GetPathHandleObjHash (const object_id_hash_t ∗ObjIdHash, uint32_t SubsysId, ns_ObjHandle_t ∗FilesetRootHandle, ns_ObjHandle_t ∗ObjHandle, char ∗Path)

    Retrieves fileset, object handle, and path information using an object and subsystem.
- int hpss_GetPathObjHash (const object_id_hash_t ∗ObjIdHash, uint32_t SubsysId, ns_ObjHandle_t ∗Fileset-RootHandle, char ∗Path)

    Retrieves the pathname and root fileset that correspond to an Object and Subsystem.

9.15.1 Detailed Description

Functions used to manipulate or construct HPSS objects.

9.15.2 Function Documentation

9.15.2.1 int hpss_GetObjFromHandle (const **ns_ObjHandle_t** ∗ *ObjHandle,* **object_id_hash_t** ∗ *ObjHash*)

Retrieves an object from a handle.

This routine returns the object id given the object handle.

Parameters

in	*ObjHandle*	parent object handle
in,out	*ObjHash*	object id / hash pair

Return values

HPSS_E_NOERROR	Success
-EFAULT	The ObjHandle parameter is a NULL pointer.

9.15.2.2 uint32_t hpss_GetObjType (const **ns_ObjHandle_t** ∗ *ObjHandle*)

Retrieves the object type given its handle.

This routine returns the object type given the object handle.

Parameters

`in`	*ObjHandle*	parent object handle

Return values

-EFAULT	The ObjHandle parameter is a NULL pointer.
-EINVAL	The ObjHandle points to an invalid object.
other	Object type

9.15.2.3 int hpss_GetPathHandle (const ns_ObjHandle_t ∗ *ObjHandle,* ns_ObjHandle_t ∗ *FilesetRootHandle,* char ∗ *Path*)

Retrieves the pathname and root fileset that correspond with an Objhandle.

The 'hpss_GetPathHandle' function outputs a pathname and a root fileset handle that corresponds to an ObjHandle.

Parameters

`in`	*ObjHandle*	object handle
`out`	*FilesetRoot-Handle*	Fileset root handle
`out`	*Path*	Path to object

Return values

-EFAULT	The Path or FilesetRootHandle parameter is NULL.
-EINVAL	The ObjHandle is NULL.

9.15.2.4 int hpss_GetPathHandleObjHash (const object_id_hash_t ∗ *ObjIdHash,* uint32_t *SubsysId,* ns_ObjHandle_t ∗ *FilesetRootHandle,* ns_ObjHandle_t ∗ *ObjHandle,* char ∗ *Path*)

Retrieves fileset, object handle, and path information using an object and subsystem.

The 'hpss_GetPathHandleObjHash' function outputs an object handle, a pathname and a root fileset handle that corresponds to an Object Id / Object Hash and Subsystem Id.

Parameters

`in`	*ObjIdHash*	object hash
`in`	*SubsysId*	subsystem id
`out`	*FilesetRoot-Handle*	Fileset root handle
`out`	*ObjHandle*	object handle
`out`	*Path*	Path to object

Return values

0	Success
-EFAULT	Path pointer is NULL
-ENAMETOOLONG	The path name could not be retrieved because it is too long.
-ENOENT	The path could not be retrieved because no path

9.15.2.5 int hpss_GetPathObjHash (const object_id_hash_t * *ObjIdHash,* uint32_t *SubsysId,* ns_ObjHandle_t * *FilesetRootHandle,* char * *Path*)

Retrieves the pathname and root fileset that correspond to an Object and Subsystem.

The 'hpss_GetPathObjHash' function outputs a pathname and a root fileset handle that corresponds to an Object Id and Subsystem Id.

Parameters

in	*ObjIdHash*	object id / hash pair
in	*SubsysId*	subsystem id
out	*FilesetRoot-Handle*	Fileset root handle
out	*Path*	Path to object

Return values

-EFAULT	Path pointer is NULL
-ENAMETOOLONG	The path name could not be retrieved because it is too long.
-ENOENT	The path could not be retrieved because no path corresponds to the given object id.

9.16 Buffered Data

Functions used to read and write data using a stream buffer.

Functions

- int hpss_Fclose (HPSS_FILE ∗hpss_Stream)

  Closes a stream and cleans up.
- int hpss_Fcntl (int hpss_fd, int Cmd, long Arg)

  Set or get file control arguments.
- int hpss_Fflush (HPSS_FILE ∗hpss_Stream)

  Writes buffered data for a stream.
- int hpss_Fgetc (HPSS_FILE ∗stream)

  Retrieve a character from the stream.
- char ∗ hpss_Fgets (char ∗s, int n, HPSS_FILE ∗stream)

  Read a string from a stream buffer.
- HPSS_FILE ∗ hpss_Fopen (const char ∗Path, const char ∗Mode)

  Opens an HPSS file and associates a stream with it.
- size_t hpss_Fread (void ∗Ptr, size_t Size, size_t Num, HPSS_FILE ∗hpss_Stream)

  Provides a buffered I/O front-end to the hpss_Read function.
- int hpss_Fseek (HPSS_FILE ∗hpss_Stream, int64_t OffsetIn, int Whence)

  Seek to an offset within the stream.
- int hpss_Fsync (int hpss_fd)

  Checks that a file descriptor is valid.
- long hpss_Ftell (HPSS_FILE ∗hpss_Stream)

  Retrieve the current offset of the stream.
- size_t hpss_Fwrite (const void ∗Ptr, size_t Size, size_t Num, HPSS_FILE ∗hpss_Stream)

  Provides a buffered I/O front-end to hpss_Write.

9.16.1 Detailed Description

Functions used to read and write data using a stream buffer.

9.16.2 Function Documentation

9.16.2.1 int hpss_Fclose (HPSS_FILE ∗ *hpss_Stream*)

Closes a stream and cleans up.

Parameters

`in,out`	*hpss_Stream*	HPSS stream to be closed

The ’hpss_Fclose’ function flushes buffered data to the stream specified by the hpss_Stream structure, and then closes the stream and releases allocated resources.

Return values

0	No error. The stream is closed.
-EACCES	Permision denied.
-EBADF	The file descriptor underlying the Stream is not valid.
-EBUSY	Some other thread is manipulating this file.
-EINVAL	Invalid (NULL) hpss_Stream parameter.
-EIO	An internal error occurred.

Note

Resources Used: Memory for I/O buffers and HPSS_FILE structure.

See Also

hpss_Fflush

9.16.2.2 int hpss_Fcntl (int *hpss_fd,* int *Cmd,* long *Arg*)

Set or get file control arguments.

Parameters

`in`	*hpss_fd*	HPSS file descriptor
`in`	*Cmd*	Requested command
`in`	*Arg*	Argument to command

This function will accept the following commands in the Cmd argument and perform the indicated action:

- F_GETFL - Get the O_NONBLOCK flag for an open file.
- F_SETFL - Set the O_NONBLOCK flag for an open file.
- F_HPSS_SET_COS - Set the the class of service for the open file.
- F_HPSS_SET_FILE_SIZE - Set the open HPSS file to the appropriate COS based on size of file given by 'Arg'.

Note

- hpss_fd - A non-negative integer specifying an open HPSS file.
- Cmd - The input command to be invoked (see above).
- Arg - The argument of the 'Cmd' parameter:
- F_GETFL/F_SETFL - N/A
- F_HPSS_SET_COS - The COS identifier
- F_HPSS_SET_FILE_SIZE - Size of file in bytes.

Returns

If the F_GETFL/F_SETFL command is successful, the file's O_NONBLOCK flag will be returned. For the other commands, a zero will be returned if the command is successful, else a negative value will be returned indicating the error encountered:

Return values

-EBUSY	Filetable entry busy
-EINVAL	Invalid argument value
-EIO	Internal I/O error
-EFAULT	Invalid object handle
-ENOTSUP	Command not supported
-EBADF	Bad file descriptor
-ENOMEM	Memory allocation failed

Note

The open HPSS file must have a length of zero (empty) for the F_HPSS_SET_COS and F_HPSS_SET_-FILE_SIZE commands. Setting the O_NONBLOCK Flag for HPSS is setting the BFS_OPEN_NO_STAGE flag.

9.16.2.3 int hpss_Fflush (HPSS_FILE ∗ *hpss_Stream*)

Writes buffered data for a stream.

Parameters

`in,out`	*hpss_Stream*	Pointer to HPSS I/O Stream

The 'hpss_Fflush' function writes any buffered data for the stream specified by the hpss_Stream parameter and leaves the stream open.

Return values

0	No error. The buffer was flushed.
-EACCES	Permission denied.
-EBADF	The file descriptor underlying the Stream is not valid.
-EINVAL	Invalid (NULL) hpss_Stream parameter.
-ESTALE	Stale connections
-EIO	An internal error occurred.

See Also

hpss_Fopen hpss_Fclose

9.16.2.4 int hpss_Fgetc (HPSS_FILE ∗ *stream*)

Retrieve a character from the stream.

The 'hpss_Fgetc' function attempts to read 1 byte from a file for open handle, *'stream'* and return it to the caller. Errno will be set if there is an error.

Parameters

`in,out`	*stream*	stream from which to receive data

Return values

Positive	Character read from the data stream
EOF	End of file reached.
-EBADF	Invalid file descriptor or file not open.
-EINVAL	Invalid input parameter (Ptr Null or Size zero)
-EFAULT	NULL stream pointer.
-EBUSY	HPSS file table entry busy.
-ESTALE	Stale Connection.
-EPERM	File opened for write only or trying to read after a write without a file positioning operation.

Note

The behavior of this function is modeled after the fgetc routine.

9.16.2.5 char∗ hpss_Fgets (char ∗ *s*, int *n*, HPSS_FILE ∗ *stream*)

Read a string from a stream buffer.

The 'hpss_Fgets' function attempts to read *n* bytes from a file from open handle, *stream*, into the client buffer pointed to by *s*. The buffer is assumed to have at least one extra byte for the null termination. Errno will be set if there is an error.

Parameters

`in,out`	*s*	Buffer in which to receive data
`in`	*n*	number of bytes to read
`in`	*stream*	stream from which to receive data

Return values

Not	Null String Data In Buffer
Null	String End of File or Error Reached
EBUSY	Errno may be set to EBUSY if the HPSS file table entry is busy.
ESTALE	Errno may be set to ESTALE if the connection to HPSS has become stale.
EPERM	Errno may be set to EPERM if the file was opened for write only or if the read was attempted after a write without repositioning.
EIO	Errno may be set to EIO if more than n bytes were read or some other error occurred.

See Also

hpss_Fgetc
hpss_Fread
hpss_Read

Note

The behavior of this function is modeled after the system fgets routine.

9.16.2.6 HPSS_FILE∗ hpss_Fopen (const char ∗ *Path*, const char ∗ *Mode*)

Opens an HPSS file and associates a stream with it.

Parameters

in	*Path*	Path/name of file
in	*Mode*	Desired file mode

This function will open an HPSS file pointed to by the *Path* parameter pointer, and associate a stream with it. Return a pointer to an HPSS_FILE structure. The pointer can be used with any of the hpss stream functions which require a stream.

Note

- path - A character string specifying the path/name of the file.
- mode - File mode (r,w,a,r+,w+,a+).

Returns

If the open command is successful, a pointer to the HPSS_FILE structure will be returned. Otherwise, a NULL pointer will be returned and the system errno value will be set to indicate the error.

Note

If an error is encountered, a NULL pointer will be returned and the system errno value will be set to indicate the error encountered:

Return values

EFAULT	Path parameter is NULL
EINVAL	Invalid mode parameter
EIO	Internal error
ENOENT	Path parameter points to an empty string
ENOMEM	Memory allocation failure
EMFILE	Too many open files

Note

If the file is open for update, an output operation cannot be directly followed by an input unless an intervening hpss_Fflush() call is made. Also, an input operation cannot be directly followed by an output operation unless an intervening hpss_Fflush() call is made except when the input encounters an EOF.

Following is a list of valid values for the mode parameter and their effects on the HPSS stream:

- r - Open text file for reading. The stream is positioned at the beginning of the file.
- r+ - Open for reading and writing. The stream is positioned at the beginning of the file.
- w - Truncate the file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
- w+ - Open for reading and writing. The file is created if it doesn't exist, otherwise it is truncated to zero length. The stream is positioned at the beginning of the file.
- a - Open for appending (writing at end of file). The file is created if it doesn't exist. The stream is positioned at the end of the file.
- a+ - Open for reading and appending (writing at end of file). The file is created if it does not exist. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the file.

9.16.2.7 size_t hpss_Fread (void ∗ *Ptr,* size_t *Size,* size_t *Num,* HPSS_FILE ∗ *hpss_Stream*)

Provides a buffered I/O front-end to the hpss_Read function.

Parameters

`in`	*Ptr*	Pointer to input array.
`in`	*Size*	Size in bytes of data items
`in`	*Num*	Number of data items to read
`in,out`	*hpss_Stream*	Pointer to input stream

The hpss_Fread function will provide a buffered I/O front-end to the hpss_Read function. It will support both 32-bit and 64-bit size_t values depending on how the library is built.

The function will copy the number of data items specified by the 'Num' parameter into an array pointed to by the 'Ptr' parameter from the input stream hpss_Stream. Each data item consists of the number of bytes specified by the 'Size' parameter.

The function will stop copying bytes if an end-of-file (EOF) or error condition is encountered while reading from the input specified by the 'hpss_Stream' parameter, or the number of data items specified by the 'Num' parameter have been copied.

Returns

The function returns the number of data items actually copied or zero if the stream is at EOF or an error is encountered.

Note

Resources Used: Memory for hpss_Stream structure and I/O buffers.
Assumptions: This function's behavior is modeled after the system fread() function.
If an error is encountered, the system global errno variable will be set to indicate the error. The following values may be returned:

Return values

0	Successful read operation.
EACCES	Permission denied.
EBADF	The file descriptor underlying the stream is not valid.
EINVAL	NULL hpss_Stream parameter.
EIO	An internal HPSS error occured.
ENOENT	File doesn't exist for reading.
ENOMEM	Memory allocation failure.

9.16.2.8 int hpss_Fseek (HPSS_FILE * *hpss_Stream*, int64_t *OffsetIn*, int *Whence*)

Seek to an offset within the stream.

This function sets the file position of the open HPSS file

Parameters

`in`	*hpss_Stream*	Pointer to HPSS_FILE
`in`	*OffsetIn*	Bytes to add/subtract
`in`	*Whence*	Origin for the seek • SEEK_SET - File position set to OffsetIn from beginning of file. • SEEK_CUR - File position adjusted by OffsetIn from current file position. • SEEK_END - File position set to file size + OffsetIn

Returns

On successful completion, a value of zero will be returned. Otherwise, a negative value will be returned that indicates the error. The global errno value will also be set to the positive value.

Return values

-EINVAL	Invalid input parameter
-EFAULT	Invalid or NULL pointer
-EBADF	HPSS File not opened
-EBUSY	HPSS File table entry busy
-ESTALE	Connection has gone stale

Note

The behavior of this routine is modeled after the system fseek()

9.16.2.9 int hpss_Fsync (int *hpss_fd*)

Checks that a file descriptor is valid.

This function is a no-op and will return a zero value as long as hpss_fd is a valid hpss file descriptor.

Parameters

`in`	*hpss_fd*	valid HPSS file descriptor

Return values

0	hpss_fd is a valid file descriptor.
-EBADF	hpss_fd is not a valid file descriptor.

9.16.2.10 long hpss_Ftell (HPSS_FILE ∗ *hpss_Stream*)

Retrieve the current offset of the stream.

This function retrieves the current offset of the stream.

Parameters

`in`	*hpss_Stream*	the open HPSS stream pointer

Return values

0	Success, current position returned
-EINVAL	Invalid input parameter
-EFAULT	Invalid or NULL pointer
-EBADF	HPSS File not opened
-EBUSY	HPSS File table entry busy
-ESTALE	Connection has gone stale

9.16.2.11 size_t hpss_Fwrite (const void ∗ *Ptr,* size_t *Size,* size_t *Num,* HPSS_FILE ∗ *hpss_Stream*)

Provides a buffered I/O front-end to hpss_Write.

Parameters

`in`	*Ptr*	Pointer to input array.
`in`	*Size*	Size in bytes of data items
`in`	*Num*	Number of data items to write
`in,out`	*hpss_Stream*	Pointer to output stream

The hpss_Fwrite function will provide a buffered I/O front-end to the hpss_Write function. It will support both 32-bit and 64-bit size_t values depending on how the library is built.

The function will write the number of data items specified by the 'Num' parameter from an array pointed to by the 'Ptr' parameter to the input stream hpss_Stream. Each data item consists of the number of bytes specified by the 'Size' parameter.

The function will stop writing bytes if an error condition is encountered or the number of data items specified by the 'Num' parameter have been written.

Returns

The function returns the number of data items actually transferred.

Note

Memory for hpss_Stream structure and I/O buffers.
This function's behavior is modeled after the system fwrite() function.
If an error is encountered, the errno variable will be set to indicate the error. The following values may be returned:

Return values

0	Successful write operation.
-EACCES	Permission denied.
-EBADF	The file descriptor underlying the stream is not valid.
-EBUSY	Another thread is accessing this table entry.
-EFBIG	Size of write request $>$ 2GB
-EINVAL	NULL hpss_Stream parameter.
-EIO	An internal HPSS error occured.
-ENOENT	File doesn't exist for writing.
-ENOMEM	Memory allocation failure.
-ESTALE	Connection not valid for this file table entry.

9.17 User-defined Attributes

Functions used to update and access user-defined metadata.

Functions

- int hpss_UserAttrCloseCursor (int SubsystemId, const hpss_cursor_id_t ∗CursorId)

  Close a cursor.
- int hpss_UserAttrDeleteAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, const hpss_userattr_list_t ∗Attrs, const char ∗Schema)

  Delete attributes from a namespace path using a handle.
- int hpss_UserAttrDeleteAttrs (const char ∗Path, const hpss_userattr_list_t ∗Attrs, const char ∗Schema)

  Delete User-defined Attributes on a namespace path.
- int hpss_UserAttrGetAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, hpss_userattr_list_t ∗Attrs, int XMLFlag, int XMLSize)

  Retrieve User-defined Attributes on a namespace path using a handle.
- int hpss_UserAttrGetAttrs (const char ∗Path, hpss_userattr_list_t ∗Attrs, int XMLFlag, int XMLSize)

  Retrieve User-defined Attributes on a namespace path.
- int hpss_UserAttrGetObj (int SubsystemId, const hpss_userattr_list_t ∗Attrs, int32_t MaxObjs, object_id_-hash_list_t ∗ObjectList, hpss_cursor_id_t ∗CursorId)

  Retrieve object id / hash pairs from a subsystem which match the attribute.
- int hpss_UserAttrGetObjQuery (int SubsystemId, int32_t MaxObjs, object_id_hash_list_t ∗ObjectList, const char ∗Query, hpss_cursor_id_t ∗CursorId)

  Retrieve object id / hash pairs which match an xquery.
- int hpss_UserAttrGetXML (int SubsystemId, int32_t MaxXMLs, hpss_string_list_t ∗XML, const char ∗XQuery, hpss_cursor_id_t ∗CursorId, int XMLSize)

  Retrieve XML strings from the subsystem which match the attribute.
- int hpss_UserAttrGetXMLObj (int SubsystemId, int32_t MaxXMLObjs, object_id_hash_list_t ∗ObjectList, hpss_string_list_t ∗XML, const char ∗XPathWhere, const char ∗XPathFind, hpss_cursor_id_t ∗CursorId, int XMLSize)

  Retrieve object / XML pairs whitch match the query.
- int hpss_UserAttrListAttrs (const char ∗Path, hpss_userattr_list_t ∗Attrs, int Flags, int XMLSize)

  List User-defined Attributes associated with a namespace path.
- int hpss_UserAttrReadObj (int SubsystemId, object_id_hash_list_t ∗ObjectList, hpss_cursor_id_t ∗CursorId)

  Read objects from a previous object search.
- int hpss_UserAttrReadXML (int SubsystemId, hpss_string_list_t ∗XML, hpss_cursor_id_t ∗CursorId)

  Read object ids from a previous XML search.
- int hpss_UserAttrReadXMLObj (int SubsystemId, object_id_hash_list_t ∗ObjectList, hpss_string_list_t ∗XML, hpss_cursor_id_t ∗CursorId)

  Read more XML/objects from a previous XML/Object search.
- int hpss_UserAttrSetAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, const hpss_userattr_list_t ∗Attrs, const char ∗Schema)

  Set User-defined Attributes on a namespace path using a handle.
- int hpss_UserAttrSetAttrs (const char ∗Path, const hpss_userattr_list_t ∗Attrs, const char ∗Schema)

  Set User-defined Attributes on a namespace path.
- int hpss_UserAttrXQueryGet (const char ∗Path, const char ∗XQuery, char ∗Buffer, int XMLSize)

  Retrieve User-defined Attribute data for a path using XQuery.
- int hpss_UserAttrXQueryGetHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_-t ∗Ucred, const char ∗XQuery, char ∗Buffer, int XMLSize)

  Retrieve User-defined Attribute data for a path using XQuery and an object handle.
- int hpss_UserAttrXQueryUpdate (const char ∗Path, const char ∗XQuery, const char ∗Schema)

Update User-defined Attributes on a path using an XQuery string.

- int hpss_UserAttrXQueryUpdateHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, const char ∗XQuery, const char ∗Schema)

  Update User-defined Attributes on a path using an XQuery string and a handle.

9.17.1 Detailed Description

Functions used to update and access user-defined metadata.

9.17.2 Function Documentation

9.17.2.1 int hpss_UserAttrCloseCursor (int *SubsystemId,* const **hpss_cursor_id_t** ∗ *CursorId*)

Close a cursor.

Parameters

`in`	*SubsystemId*	Subsystem Id
`in`	*CursorId*	CursorId

Close the given cursor. This will release the core server resources allocated for this cursor and finish the transaction.

Return values

-EPERM	The user who requested to close a cursor does not match the user who initiated the operation.
-EFAULT	The CursorId parameter is NULL.
-ESRCH	The cursor requested to be closed could not be found and may have already been closed due to a timeout.

9.17.2.2 int hpss_UserAttrDeleteAttrHandle (const **ns_ObjHandle_t** ∗ *Obj,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* const **hpss_userattr_list_t** ∗ *Attrs,* const char ∗ *Schema*)

Delete attributes from a namespace path using a handle.

Parameters

`in`	*Obj*	Parent Object
`in`	*Path*	Namespace Path
`in`	*Ucred*	user credentials
`in`	*Attrs*	Attributes
`in`	*Schema*	Schema

Delete Attributes on Handle and Path requested by the Key field in Attrs. If a schema is provided, the generated XQuery will include XML validation on the provided schema.

Return values

-EACCES	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	One of the attribute pairs in Attr is invalid.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

-ENOENT	The named object does not exist, or the Path argument points to an empty string.

Note

The schema variable is not null-checked because a NULL schema indicates that no validation should be done.

9.17.2.3 int hpss_UserAttrDeleteAttrs (const char ∗ *Path*, const hpss_userattr_list_t ∗ *Attrs*, const char ∗ *Schema*)

Delete User-defined Attributes on a namespace path.

Parameters

`in`	*Path*	Namespace Path
`in`	*Attrs*	Attributes
`in`	*Schema*	Schema

Delete Attributes on Path requested by the Key field in Attrs. Schema can be provided to enforce validation on a specific schema.

Return values

-EACCES	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	One of the attribute pairs in Attr is invalid.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.

Note

The schema variable is not null-checked because a NULL schema indicates that no validation should be done.

This function will not return an error if the requested attribute does not exist; deleting an XML object is not considered an error.

9.17.2.4 int hpss_UserAttrGetAttrHandle (const ns_ObjHandle_t ∗ *Obj*, const char ∗ *Path*, const sec_cred_t ∗ *Ucred*, hpss_userattr_list_t ∗ *Attrs*, int *XMLFlag*, int *XMLSize*)

Retrieve User-defined Attributes on a namespace path using a handle.

Parameters

`in`	*Obj*	Parent Object
`in`	*Path*	Namespace Path
`in`	*Ucred*	user credentials
`out`	*Attrs*	Attributes and Values
`in`	*XMLFlag*	XML Flag
`in`	*XMLSize*	XML Size

Get Attributes on Handle and Path requested by the Key field in Attrs. XMLFlag represents whether the returning value will be XML or text only. The XML return type can be used to obtain all of the XML under and including the given path.

Return values

-EACCES	The user does not have permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer and Obj is NULL.
-EINVAL	One of the attribute pairs in Attr is invalid.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.
-EACCES	The user does not have write permission on the namepsace object, or search permission is denied on a component of the path prefix.
-ERANGE	The length of the data was too large to be returned completely.
-EDOM	The XMLSize was too large.

9.17.2.5 int hpss_UserAttrGetAttrs (const char ∗ *Path*, hpss_userattr_list_t ∗ *Attrs*, int *XMLFlag*, int *XMLSize*)

Retrieve User-defined Attributes on a namespace path.

Parameters

`in`	*Path*	Namespace Path
`out`	*Attrs*	Attributes and Values
`in`	*XMLFlag*	XML Flag
`in`	*XMLSize*	XML Size

Get Attributes on Path requested by the Key field in Attrs. XMLFlag represents whether the returning value will be XML or text only. The XML return type can be used to obtain all of the XML under and including the given path.

Return values

-EACCES	The user does not have read permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	One of the attribute pairs in Attr is invalid, or an invalid XMLFlag was used.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.
-ERANGE	The length of the data was too large to be returned completely.
-EDOM	The XMLSize was too large.

9.17.2.6 int hpss_UserAttrGetObj (int *SubsystemId*, const hpss_userattr_list_t ∗ *Attrs*, int32_t *MaxObjs*, object_id_hash_list_t ∗ *ObjectList*, hpss_cursor_id_t ∗ *CursorId*)

Retrieve object id / hash pairs from a subsystem which match the attribute.

Parameters

`in`	*SubsystemId*	Subsystem Id
`in`	*Attrs*	Attribute names
`in`	*MaxObjs*	Max objs to return
`in,out`	*ObjectList*	List of Id / Hash Pairs
`in,out`	*CursorId*	Optional CursorId

Get up to MaxObjs Objects from SubsystemId containing Attribute or Attribute/Value pair field in Attrs. If a CursorId is requested, one will be returned which can be used to read additional entries.

Return values

-EACCES	The user is not a Trusted User with Control Permission in the Core Server.
-EFAULT	The Attrs or ObjectList parameters are NULL.
-EINVAL	MaxObjs is an invalid number.
-ENOENT	There were no entries which matched the query.
-ERANGE	One or more of the results contained more data than could be returned.

Note

Cursors used by this function will expire after a certain amount of time if they are idle.

9.17.2.7 int hpss_UserAttrGetObjQuery (int *SubsystemId,* int32_t *MaxObjs,* object_id_hash_list_t ∗ *ObjectList,* const char ∗ *Query,* hpss_cursor_id_t ∗ *CursorId*)

Retrieve object id / hash pairs which match an xquery.

Parameters

`in`	*SubsystemId*	Subsystem Id
`in`	*MaxObjs*	Max objs to return
`out`	*ObjectList*	Objects Found
`in`	*Query*	Query to Execute
`in,out`	*CursorId*	Cursor ID

Execute Query on subsystem ID. The given query will be a valid XPath string. It will return a list of objects (up to MaxObjs) which match the query. If requested, a cursor Id will be returned which will enable additional reads on that cursor.

Note

Cursors used by this function will expire after a certain amount of time if they are idle.

Return values

-EACCES	The user is not a Trusted User with Control Permission in the Core Server.
-EFAULT	The XQuery or ObjectList parameters are NULL.
-EINVAL	MaxObjs is an invalid number; XQuery is invalid.
-ENOENT	There were no entries which matched the query.
-ERANGE	One or more results contained more data than could be returned.

9.17.2.8 int hpss_UserAttrGetXML (int *SubsystemId,* int32_t *MaxXMLs,* hpss_string_list_t ∗ *XML,* const char ∗ *XQuery,* hpss_cursor_id_t ∗ *CursorId,* int *XMLSize*)

Retrieve XML strings from the subsystem which match the attribute.

Parameters

`in`	*SubsystemId*	Subsystem Id
`in`	*MaxXMLs*	Max Entries to Return
`out`	*XML*	XML Returned
`in`	*XQuery*	XQuery

`out`	*CursorId*	Optional CursorId
`in`	*XMLSize*	XML Size

Get up to MaxXMLs Objects from SubsystemId containing Attributes or Attribute/Value pairs field in Attrs. If a CursorId is requested, one will be returned which can be used to read additional entries.

Return values

-EACCES	The user is not a Trusted User with Control Permission in the Core Server.
-EFAULT	The XQuery or XML parameters are NULL.
-EINVAL	MaxXMLs is an invalid number; XQuery is invalid.
-ENOENT	There were no entries which matched the query.
-ERANGE	One or more of the results contained more data than could be returned.
-EDOM	The XMLSize was too large.

Note

Cursors used by this function will expire after a certain amount of time if they are idle.

The XML string structure must be freed by the caller.

9.17.2.9 int hpss_UserAttrGetXMLObj (int *SubsystemId,* int32_t *MaxXMLObjs,* object_id_hash_list_t * *ObjectList,* hpss_string_list_t * *XML,* const char * *XPathWhere,* const char * *XPathFind,* hpss_cursor_id_t * *CursorId,* int *XMLSize*)

Retrieve object / XML pairs whitch match the query.

Get Object / XML pairs, up to MaxXMLObjs. These pairs will match the XPath contained in XPathWhere, and the XML portion will contain results from XPathFind. Both the attribute(s) described by XPathWhere and XPathFind must exist for a value to be retrieved; no nulls will be returned. If requested, a CursorId will be returned which can be used to retrieve additional results.

Parameters

`in`	*SubsystemId*	Subsystem Id
`in`	*MaxXMLObjs*	Max XMLObjs to return
`out`	*ObjectList*	Object List
`out`	*XML*	XML List
`in`	*XPathWhere*	'Where' XPath
`in`	*XPathFind*	'Find' XPath
`in,out`	*CursorId*	CursorId
`in`	*XMLSize*	XML Size

Note

Cursors used by this function will expire after a certain amount of time if they are idle.

Return values

-EACCES	The user is not a Trusted User with Control Permission in the Core Server.
-EFAULT	The XQuery, ObjectList, or XML parameters are NULL.
-EINVAL	MaxXMLObjs is an invalid number; XPathFind or XPathWhere is invalid.
-ENOENT	There were no entries which matched the query.

-ERANGE	One or more of the results contained more data than could be returned.

9.17.2.10 int hpss_UserAttrListAttrs (const char ∗ *Path*, hpss_userattr_list_t ∗ *Attrs*, int *Flags*, int *XMLSize*)

List User-defined Attributes associated with a namespace path.

Parameters

`in`	*Path*	Namespace Path
`out`	*Attrs*	Attributes
`in`	*Flags*	Flags
`in`	*XMLSize*	XML Size

Obtain all attributes associated with Path. Additionally, this function will put occurrence identifiers next to attributes with multiple occurrences, and obtain XML element attributes. The results are guaranteed to be in the order they appear in the XML document.

Return values

0	Success
-EACCES	The user does not have read permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	The XML from Path could not be converted into Attribute format.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.
-ERANGE	The length of the data requested was too large to be returned.
-EDOM	The XMLSize was too large.

Warning

The attributes returned by this function must be freed by the caller.

9.17.2.11 int hpss_UserAttrReadObj (int *SubsystemId*, object_id_hash_list_t ∗ *ObjectList*, hpss_cursor_id_t ∗ *CursorId*)

Read objects from a previous object search.

Read additional Objects from a previous call to a GetObjs function. When the last set of results come back, the cursor will be invalidated and the transaction and the cursor entry will be removed. If no entries are read for a certain amount of time defined by the Core Server environment, the transaction and cursor entry will be cleaned up and removed prematurely. The client can close the cursor manually by calling hpss_UserAttrCloseCursor.

Parameters

`in`	*SubsystemId*	Subsystem Id
`out`	*ObjectList*	Objects
`in`	*CursorId*	CursorId

Return values

-EACCES	The user requested a read from this cursor does not match the user who initiated the operation.
-EFAULT	The ObjectList or CursorId parameters are NULL.
-ENOENT	There were no entries left to read.
-ESRCH	The cursor requested for reading was not found; it was likely invalidated due to prolonged inactivity.
-ERANGE	One or more of the results contained more data than could be returned.

9.17.2.12 int hpss_UserAttrReadXML (int *SubsystemId,* hpss_string_list_t ∗ *XML,* hpss_cursor_id_t ∗ *CursorId*)

Read object ids from a previous XML search.

Parameters

`in`	*SubsystemId*	Subsystem Id
`out`	*XML*	XML Returned
`in`	*CursorId*	Cursor Id

Read additional XML from a previous call to a GetXMLs function. The buffer size must be the same as the previous call. When the last set of results come back, the cursor will be invalid and the transaction and the cursor entry will be removed.

Return values

-EACCES	The user requested a read from this cursor does not match the user who initiated the operation.
-EFAULT	The XML or CursorId parameters are NULL.
-ENOENT	There were no entries left to read.
-ESRCH	The cursor requested for reading was not found; it was likely invalidated due to prolonged inactivity.
-ERANGE	One or more of the results contained more data than could be returned.

9.17.2.13 int hpss_UserAttrReadXMLObj (int *SubsystemId,* object_id_hash_list_t ∗ *ObjectList,* hpss_string_list_t ∗ *XML,* hpss_cursor_id_t ∗ *CursorId*)

Read more XML/objects from a previous XML/Object search.

Parameters

`in`	*SubsystemId*	Subsystem Id
`out`	*ObjectList*	Objects
`out`	*XML*	XML Returned
`in`	*CursorId*	Cursor Id

Read additional XML/Objects from a previous call to a GetXMLObjs function; the buffer size must be the same as the previous call. When the last set of results come back, the cursor will be invalid and the transaction and the cursor entry will be removed.

Return values

-EACCES	The user requested a read from this cursor does not match the user who initiated the operation.
-EFAULT	The ObjectList, XML, or CursorId parameters are NULL.
-ENOENT	There were no entries left to read.

-ESRCH	The cursor requested for reading was not found; it was likely invalidated due to prolonged inactivity.
-ERANGE	One or more of the results contained more data than could be retrieved.

9.17.2.14 int hpss_UserAttrSetAttrHandle (const ns_ObjHandle_t * *Obj*, const char * *Path*, const sec_cred_t * *Ucred*, const hpss_userattr_list_t * *Attrs*, const char * *Schema*)

Set User-defined Attributes on a namespace path using a handle.

Set Attributes on Handle and Path with the (XPath,Value) pairs in Attrs. If Schema is supplied, the input XML will be validated by the database against the named Schema.

Parameters

`in`	*Obj*	Parent Object
`in`	*Path*	Namespace Path
`in`	*Ucred*	user credentials
`in`	*Attrs*	Attributes
`in`	*Schema*	Schema

Return values

-EACCES	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer and Obj is NULL.
-EINVAL	One of the attribute pairs in Attr is invalid.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.

Note

The schema variable is not null-checked because a NULL schema indicates that no validation should be done.

9.17.2.15 int hpss_UserAttrSetAttrs (const char * *Path*, const hpss_userattr_list_t * *Attrs*, const char * *Schema*)

Set User-defined Attributes on a namespace path.

Parameters

`in`	*Path*	Namespace Path
`in`	*Attrs*	Attributes
`in`	*Schema*	Schema

Set Attributes on Path with the (XPath,Value) pairs in Attrs. If Schema is supplied, the input XML will be validated by the database against the named Schema.

Note

The schema variable is not null-checked because a NULL schema indicates that no validation should be done.

Return values

0	Success
-EACCES	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path or Attrs parameter is a NULL pointer.
-EINVAL	One of the attribute pairs in Attr is invalid.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.

9.17.2.16 int hpss_UserAttrXQueryGet (const char ∗ *Path,* const char ∗ *XQuery,* char ∗ *Buffer,* int *XMLSize*)

Retrieve User-defined Attribute data for a path using XQuery.

Parameters

`in`	*Path*	Namespace Path
`in`	*XQuery*	XQuery
`in,out`	*Buffer*	Buffer
`in`	*XMLSize*	XML Size

Get XML from Path using XQuery and return the results in Buffer.

Return values

-EACCES	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	Invalid XQuery.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.
-ERANGE	The requested data was too large to be returned completely.

9.17.2.17 int hpss_UserAttrXQueryGetHandle (const ns_ObjHandle_t ∗ *Obj,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* const char ∗ *XQuery,* char ∗ *Buffer,* int *XMLSize*)

Retrieve User-defined Attribute data for a path using XQuery and an object handle.

Parameters

`in`	*Obj*	Parent Object
`in`	*Path*	Namespace Path
`in`	*Ucred*	user credentials
`in`	*XQuery*	XQuery
`in,out`	*Buffer*	Buffer
`in`	*XMLSize*	XML Size

Return values

-EACCES	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	Invalid XQuery.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.
-ERANGE	The requested data was too large to be returned completely.

9.17.2.18 int hpss_UserAttrXQueryUpdate (const char ∗ *Path,* const char ∗ *XQuery,* const char ∗ *Schema*)

Update User-defined Attributes on a path using an XQuery string.

Parameters

`in`	*Path*	Namespace Path
`in`	*XQuery*	XQuery string
`in`	*Schema*	Schema

Return values

-EACCES	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	Invalid XQuery.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.

Note

The Schema is not null-checked because if it is NULL, then no validation will be done.

9.17.2.19 int hpss_UserAttrXQueryUpdateHandle (const ns_ObjHandle_t ∗ *Obj,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* const char ∗ *XQuery,* const char ∗ *Schema*)

Update User-defined Attributes on a path using an XQuery string and a handle.

Parameters

`in`	*Obj*	Namspace Object
`in`	*Path*	Namespace Path
`in`	*Ucred*	user credentials
`in`	*XQuery*	XQuery
`in`	*Schema*	Schema

Return values

-EACCES	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.

-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	Invalid XQuery.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named object does not exist, or the Path argument points to an empty string.

Note

The Schema is not null-checked because if it is NULL, then no validation will be done.

9.18 Data Structure Conversion

Functions used to convert HPSS constructs to POSIX equivalents.

Functions

- void API_CalcBlock32 (const hpss_Attrs_t ∗Attrs, uint32_t ∗Blksize, uint32_t ∗Numblks)

  Calculates the block size and number of blocks necessary for data.
- void API_CalcBlock64 (const hpss_Attrs_t ∗Attrs, uint64_t ∗Blksize, uint64_t ∗Numblks)

  Calculates the block size and number of blocks necessary for data.
- void API_ConvertModeToPosixMode (const hpss_Attrs_t ∗Attrs, mode_t ∗PosixMode)

  Converts an input HPSS mode value to its corresponding POSIX mode value.
- void API_ConvertPosixModeToMode (mode_t PosixMode, hpss_Attrs_t ∗Attrs)

  Converts a POSIX mode value to its equivalent HPSS mode value.
- void API_ConvertPosixTimeToTime (hpss_Attrs_t ∗Attrs, timestamp_sec_t Atime, timestamp_sec_t Mtime, timestamp_sec_t Ctime)

  Converts POSIX time values to corresponding HPSS time values.
- void API_ConvertTimeToPosixTime (const hpss_Attrs_t ∗Attrs, timestamp_sec_t ∗Atime, timestamp_sec_t ∗Mtime, timestamp_sec_t ∗Ctime)

  Converts HPSS time values to corresponding POSIX mode values.

9.18.1 Detailed Description

Functions used to convert HPSS constructs to POSIX equivalents.

9.18.2 Function Documentation

9.18.2.1 void API_CalcBlock32 (const **hpss_Attrs_t** ∗ ***Attrs,*** uint32_t ∗ ***Blksize,*** uint32_t ∗ ***Numblks***)

Calculates the block size and number of blocks necessary for data.

This routine returns a block size and the number of blocks that would be required given the HPSS data length. Either output parameter can be NULL if desired.

Parameters

`in`	*Attrs*	HPSS object attributes
`out`	*Blksize*	Block Size
`out`	*Numblks*	Number of Blocks

See Also

API_CalcBlock64, API_ConvertModeToPosixMode, API_ConvertPosixModeToMode, ∗ API_ConvertPosixTimeToTime, API_ConvertTimeToPosixTime

Note

The block size and number of blocks will use 32-bit integers.

9.18.2.2 void API_CalcBlock64 (const hpss_Attrs_t ∗ *Attrs,* uint64_t ∗ *Blksize,* uint64_t ∗ *Numblks*)

Calculates the block size and number of blocks necessary for data.

This routine returns a block size and the number of blocks that would be required given the HPSS data length. Either output parameter can be NULL if desired.

Parameters

`in`	*Attrs*	HPSS object attributes
`out`	*Blksize*	Block Size
`out`	*Numblks*	Number of Blocks

See Also

API_CalcBlock32, API_ConvertModeToPosixMode, API_ConvertPosixModeToMode, API_ConvertPosixTime-ToTime, API_ConvertTimeToPosixTime

Note

The block size and number of blocks will use the HPSS 64-bit type.

9.18.2.3 void API_ConvertModeToPosixMode (const hpss_Attrs_t ∗ *Attrs,* mode_t ∗ *PosixMode*)

Converts an input HPSS mode value to its corresponding POSIX mode value.

This routine translates an HPSS attribute structure to the equivalent POSIX mode_t value.

Parameters

`in`	*Attrs*	HPSS object attributes
`out`	*PosixMode*	Pointer to posix mode_t

See Also

API_CalcBlock32, API_CalcBlock64, API_ConvertPosixModeToMode, API_ConvertPosixTimeToTime, API_-ConvertTimeToPosixTime

9.18.2.4 void API_ConvertPosixModeToMode (mode_t *PosixMode,* hpss_Attrs_t ∗ *Attrs*)

Converts a POSIX mode value to its equivalent HPSS mode value.

This routine translates a POSIX mode_t value to the equivalent HPSS namespace attributes.

Parameters

`in`	*PosixMode*	POSIX mode_t to be translated
`out`	*Attrs*	Pointer to HPSS attributes

See Also

API_CalcBlock32, API_CalcBlock64, API_ConvertModeToPosixMode API_ConvertPosixTimeToTime, API_-ConvertTimeToPosixTime

9.18.2.5 void API_ConvertPosixTimeToTime (hpss_Attrs_t ∗ *Attrs*, timestamp_sec_t *Atime*, timestamp_sec_t *Mtime*, timestamp_sec_t *Ctime*)

Converts POSIX time values to corresponding HPSS time values.

This routine translates POSIX time values to the equivalent HPSS time values.

Parameters

out	*Attrs*	HPSS object attributes
in	*Atime*	Last Access time
in	*Mtime*	Last Modify time
in	*Ctime*	Last Change time

See Also

API_CalcBlock32, API_CalcBlock64, API_ConvertModeToPosixMode API_ConvertPosixModeToMode, API_-ConvertTimeToPosixTime

9.18.2.6 void API_ConvertTimeToPosixTime (const hpss_Attrs_t ∗ *Attrs*, timestamp_sec_t ∗ *Atime*, timestamp_sec_t ∗ *Mtime*, timestamp_sec_t ∗ *Ctime*)

Converts HPSS time values to corresponding POSIX mode values.

This routine translates HPSS namespace attributes to the equivalent POSIX time values.

Parameters

in	*Attrs*	HPSS object attributes
out	*Atime*	Last Access time
out	*Mtime*	Last Modify time
out	*Ctime*	Last Change time

See Also

API_CalcBlock32, API_CalcBlock64, API_ConvertModeToPosixMode API_ConvertPosixModeToMode, API_-ConvertPosixTimeToTime

9.19 Parallel I/O

Functions which implement HPSS parallel I/O capabilities.

Functions

- int hpss_PIOEnd (hpss_pio_grp_t StripeGroup)

  End and clean up a parallel IO group context.
- int hpss_PIOExecute (int Fd, uint64_t FileOffset, uint64_t Size, hpss_pio_grp_t StripeGroup, hpss_pio_-gapinfo_t ∗GapInfo, uint64_t ∗BytesMoved)

  Begin a PIO data transfer.
- int hpss_PIOExecuteItem (int Fd, uint64_t FileOffset, uint64_t Size, const hpss_pio_grp_t StripeGroup, hpss-_pio_gapinfo_t ∗GapInfo, uint64_t ∗BytesMoved, hpss_read_queue_item_t ∗Item)

  Begin a PIO data transfer, with a reference to a running read queue item.
- int hpss_PIOExportGrp (const hpss_pio_grp_t StripeGroup, void ∗∗Buffer, unsigned int ∗BufLength)

  Export a parallel IO group context to a buffer.
- int hpss_PIOFinalize (hpss_pio_grp_t ∗StripeGroup)

  Safely end and clean up a parallel IO group context.
- int hpss_PIOGetPartError (hpss_pio_grp_t StripeGroup, int ∗Error)

  Retrieves a participant error code.
- int hpss_PIOGetRequestID (hpss_pio_grp_t StripeGroup, hpss_reqid_t ∗RequestID)

  Retrieves the request ID for a group.
- int hpss_PIOImportGrp (const void ∗Buffer, unsigned int BufLength, hpss_pio_grp_t ∗StripeGroup)

  Import a group which was exported with hpss_PIOExportGrp.
- int hpss_PIORegister (uint32_t StripeElement, const hpss_sockaddr_t ∗DataNetSockAddr, void ∗DataBuffer, uint32_t DataBufLen, hpss_pio_grp_t StripeGroup, const hpss_pio_cb_t IOCallback, const void ∗IOCallback-Arg)

  Register a PIO stripe participant.
- int hpss_PIOStart (hpss_pio_params_t ∗InputParams, hpss_pio_grp_t ∗StripeGroup)

  Start a new parallel I/O group context.

9.19.1 Detailed Description

Functions which implement HPSS parallel I/O capabilities.

9.19.2 Function Documentation

9.19.2.1 int hpss_PIOEnd (hpss_pio_grp_t *StripeGroup*)

End and clean up a parallel IO group context.

Parameters

`in,out`	*StripeGroup*	IO stripe group

This function is used by a transfer coordinator or participant to end a parallel IO group context.

Return values

-ENOMEM	Can't allocate memory for stack entry
-EINVAL	Group is NULL
0	otherwise

Note

This function runs in either the coordinator or stripe participant context/role.

Deprecated Deprecated in HPSS 7.5; use hpss_PIOFinalize() instead.

Warning

This function does not set *StripeGroup* to NULL and thus multiple calls to this function using the same StripeGroup will result in an access of a freed pointer.

9.19.2.2 int hpss_PIOExecute (int *Fd*, uint64_t *FileOffset*, uint64_t *Size*, hpss_pio_grp_t *StripeGroup*, hpss_pio_gapinfo_t ∗ *GapInfo*, uint64_t ∗ *BytesMoved*)

Begin a PIO data transfer.

Parameters

`in`	*Fd*	Open file descriptor
`in`	*FileOffset*	IO file offset
`in`	*Size*	IO size
`in`	*StripeGroup*	IO stripe group
`out`	*GapInfo*	Sparse file information
`out`	*BytesMoved*	Bytes moved

This function is used by a transfer coordinator to begin a data transfer.

Return values

0	Success
-ENOMEM	Can't allocate memory for stack entry
-EINVAL	The group context is NULL or cannot be verified.
-EINVAL	The I/O size was 0.

Note

This function runs in the coordinator context/role.

9.19.2.3 int hpss_PIOExecuteItem (int *Fd*, uint64_t *FileOffset*, uint64_t *Size*, const hpss_pio_grp_t *StripeGroup*, hpss_pio_gapinfo_t ∗ *GapInfo*, uint64_t ∗ *BytesMoved*, hpss_read_queue_item_t ∗ *Item*)

Begin a PIO data transfer, with a reference to a running read queue item.

This function behaves the exact same as hpss_PIOExecute, except that it connects the PIO request with the specified read queue item. See hpss_ReadQueueAdd for more information.

Parameters

in	*Fd*	Open file descriptor
in	*FileOffset*	IO file offset
in	*Size*	IO size
in	*StripeGroup*	IO stripe group
out	*GapInfo*	Sparse file information
out	*BytesMoved*	Bytes moved
in	*Item*	Read Queue Item

This function is used by a transfer coordinator to begin a data transfer.

Return values

0	Success
-EIO	Cannot set the request ID.
-ENOMEM	Can't allocate memory for stack entry
-EINVAL	The group context is NULL or cannot be verified.
-EINVAL	The I/O size was 0.

Note

This function runs in the coordinator context/role.

9.19.2.4 int hpss_PIOExportGrp (const hpss_pio_grp_t *StripeGroup*, void ∗∗ *Buffer*, unsigned int ∗ *BufLength*)

Export a parallel IO group context to a buffer.

Parameters

in	*StripeGroup*	IO stripe group
out	*Buffer*	Raw data buffer
out	*BufLength*	Raw buffer length

This function is used by a transfer coordinator to convert parallel IO group context data in to a raw data that is suitable to transfer across the network.

Return values

-ENOMEM	Can't allocate memory for stack entry
-EFAULT	Buffer or Buffer length were NULL.
-EINVAL	Group context was NULL or could not be verified.
0	otherwise

Note

This function runs in the coordinator context/role.

9.19.2.5 int hpss_PIOFinalize (hpss_pio_grp_t ∗ *StripeGroup*)

Safely end and clean up a parallel IO group context.

Parameters

`in`	*StripeGroup*	IO stripe group

This function is used by a transfer coordinator or participant to end a parallel IO group context. It is functionally equivalent to hpss_PIOEnd() except it guards against a scenario where hpss_PIOEnd is called twice.

Return values

-EFAULT	NULL stripe group provided
0	Success
Other	Error from hpss_PIOEnd()

Note

This function runs in either the coordinator or stripe participant context/role.
This function is a safe version of hpss_PIOEnd;it sets the stripe pointer to NULL after it is freed to avoid use after and double free issues.

9.19.2.6 int hpss_PIOGetPartError (**hpss_pio_grp_t** *StripeGroup*, int ∗ *Error*)

Retrieves a participant error code.

Parameters

`in`	*StripeGroup*	IO stripe group
`in,out`	*Error*	Participant Error Code

This function is called to retrieve the participant error code from the stripe group context.

Return values

0	Success
-EFAULT	No group supplied
-EINVAL	No error supplied

9.19.2.7 int hpss_PIOGetRequestID (**hpss_pio_grp_t** *StripeGroup*, **hpss_reqid_t** ∗ *RequestID*)

Retrieves the request ID for a group.

Parameters

`in`	*StripeGroup*	IO stripe group
`in,out`	*RequestID*	Coordinator Request ID

This function is called to retrieve the coordinator's request id from the stripe group context.

Return values

0	Success
-EFAULT	No group supplied
-EINVAL	No request ID supplied

Note

For other I/O, use hpss_GetNextIORequestID to obtain the request id.

9.19.2.8 int hpss_PIOImportGrp (const void ∗ *Buffer*, unsigned int *BufLength*, **hpss_pio_grp_t** ∗ *StripeGroup*)

Import a group which was exported with hpss_PIOExportGrp.

Parameters

`in`	*Buffer*	Raw data buffer
`in`	*BufLength*	Raw buffer length
`out`	*StripeGroup*	IO stripe group

This function is used by a transfer stripe participant to convert raw parallel IO group context data from in to data that is suitable for hpss_PIORegister()

Return values

-ENOMEM	Can't allocate memory for stack entry
-EINVAL	StripeGroup or Buffer is NULL
0	otherwise

Note

This function runs in the participant context/role.

9.19.2.9 int hpss_PIORegister (uint32_t *StripeElement*, const hpss_sockaddr_t ∗ *DataNetSockAddr*, void ∗ *DataBuffer*, uint32_t *DataBufLen*, hpss_pio_grp_t *StripeGroup*, const hpss_pio_cb_t *IOCallback*, const void ∗ *IOCallbackArg*)

Register a PIO stripe participant.

Parameters

`in`	*StripeElement*	Element of the stripe
`in`	*DataNetSock-Addr*	NIC address for data movement
`in`	*DataBuffer*	Data buffer for IO
`in`	*DataBufLen*	Data buffer length
`in,out`	*StripeGroup*	IO stripe group
`in`	*IOCallback*	IO call back function
`in`	*IOCallbackArg*	IO call back argument

This is used by a transfer stripe participant to register an IO call back function.

Return values

0	Success
-EINVAL	Invalid argument or checksum.
-ENOMEM	Not enough memory to allocate stripe group.

Note

This function runs in the participant context/role.

9.19.2.10 int hpss_PIOStart (hpss_pio_params_t ∗ *InputParams*, hpss_pio_grp_t ∗ *StripeGroup*)

Start a new parallel I/O group context.

Parameters

`in,out`	*InputParams*	Input parameters
`out`	*StripeGroup*	New IO stripe group

This function is used by a transfer coordinator to start a new parallel IO group context.

Return values

0	Success
-EFAULT	StripeGroup provided was NULL.
-EINVAL	InputParams provided was NULL, invalid operation specified, or zero block size provided.
-ENOMEM	if memory can't be allocated for stripe group

Note

This function runs in the coordinator context/role.
This function will set up certain defaults for the *InputParams* if they are not provided explicitly by the caller.

9.20 Trashcan

Functions which use the HPSS trashcan feature.

Functions

- int hpss_GetTrashcan (const char ∗Path, char ∗TrashcanBuf, int TrashcanBufLength, ns_ObjHandle_-t ∗FilesetRoot)

  Locate the trashcan for a provided path.
- int hpss_GetTrashcanHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_-t ∗Ucred, char ∗TrashcanBuf, int TrashcanBufLength, ns_ObjHandle_t ∗FilesetRoot)

  Locate the trashcan for a provided handle/path.
- int hpss_GetTrashSettings (ns_TrashcanSettings_t ∗TrashSettings)

  Query the Root Core Server for trashcan settings.
- int hpss_ReadTrash (signed32 SubsystemId, const unsigned32 ∗UserIdP, const unsigned32 ∗RealmIdP, un-signed32 BufferSize, unsigned32 ∗End, ns_TrashEntry_t ∗TrashPtr)

  Read trash entries for a subsystem.
- int hpss_Undelete (const char ∗Path, hpss_undelete_flags_t Flag)

  Restore a namespace entry from the trash.
- int hpss_UndeleteHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, hpss_undelete_flags_t Flag)

  Restore a namespace entry from the trash.

9.20.1 Detailed Description

Functions which use the HPSS trashcan feature.

9.20.2 Function Documentation

9.20.2.1 int hpss_GetTrashcan (const char ∗ *Path,* char ∗ *TrashcanBuf,* int *TrashcanBufLength,* ns_ObjHandle_t ∗ *FilesetRoot*)

Locate the trashcan for a provided path.

Parameters

in	*Path*	Path to lookup trashcan for
in,out	*TrashcanBuf*	Buffer for trashcan path
in	*TrashcanBuf-Length*	TrashcanBuf's length
in,out	*FilesetRoot*	The trashcan's fileset root

Locates a trashcan's path (relative to its fileset root) based upon the provided path. This function can also return the fileset root for the trashcan.

Return values

HPSS_E_NOERROR	Success
HPSS_EFAULT	Path or Trashcan Buffer not provided.

HPSS_ENOENT	Path does not exist
HPSS_ENOTDIR	Some component of the path is not a directory
HPSS_ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit
HPSS_EINVAL	Invalid trashcan length specified

Note

This function can be used in a couple of different ways. For one, it can be used to determine for a given object about to be deleted, which trashcan it would be placed in. Additionally it could be used on a directory to identify the trashcan that any object within that directory would be placed in (perhaps useful in caching a path to the current working dir's trashcan).
This function returns the path to the trashcan a deleted object should be soft deleted to – and it will return that path even if the actual trashcan has not been created yet.
This function returns a path to a trashcan regardless of whether trashcans are actually enabled on the server side. Getting a trashcan path back from this function should never be taken as meaning that trashcans are enabled.

9.20.2.2 int hpss_GetTrashcanHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* char ∗ *TrashcanBuf,* int *TrashcanBufLength,* ns_ObjHandle_t ∗ *FilesetRoot*)

Locate the trashcan for a provided handle/path.

Parameters

`in`	*ObjHandle*	Object Handle for lookup
`in`	*Path*	Path to lookup trashcan for
`in`	*Ucred*	User Credentials
`in,out`	*TrashcanBuf*	Buffer for trashcan path
`in`	*TrashcanBuf-Length*	TrashcanBuf's length
`in,out`	*FilesetRoot*	The trashcan's fileset root

Locates a trashcan's path (relative to its fileset root) based upon the provided handle and path. This function can also return the fileset root for the trashcan.

Return values

HPSS_E_NOERROR	Success
HPSS_ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
HPSS_EFAULT	Path or Trashcan Buffer not provided.
HPSS_ENOENT	Path does not exist
HPSS_ENOTDIR	Some component of the path is not a directory
HPSS_EINVAL	Invalid trashcan length specified or no handle provided.

Note

See hpss_GetTrashcan for additional notes about this function.

9.20.2.3 int hpss_GetTrashSettings (ns_TrashcanSettings_t ∗ *TrashSettings*)

Query the Root Core Server for trashcan settings.

The 'hpss_GetTrashSettings' function queries the root Core Server for the trashcan settings. This function can only be used by an HPSS authorized caller.

Parameters

out	*TrashSettings*	Trashcan Settings

Return values

0	No error. Trash settings in the buffer.
-EFAULT	The TrashSettings was null.
-EACCES	The user does not have authority to get this information.

Note

The trash settings can be modified by the admin (or disabled) at any time; the trash settings should no be assumed to reflect a constant state.

9.20.2.4 int hpss_ReadTrash (signed32 *SubsystemId,* const unsigned32 ∗ *UserIdP,* const unsigned32 ∗ *RealmIdP,* unsigned32 *BufferSize,* unsigned32 ∗ *End,* ns_TrashEntry_t ∗ *TrashPtr*)

Read trash entries for a subsystem.

The 'hpss_ReadTrash' function returns an array of trash records, TrashPtr, representing items found in the user's trashcans on the subsystem SubsystemId. The number of entries is limited by BufferSize, which describes the size of the allocated TrashPtr structure.

hpss_ReadTrash behaves like hpss_ReadAttrsPlus(), but with a few key differences. Upon the first call, the trashcan contents are queried and an initial buffer returned. Subsequent calls will return additional trashcan items until the end of the trashcan contents is reached. When all records have been read, End will be set to TRUE. In reading the trash, there is no capability to request a specific offset or to rewind to prior results.

hpss_ReadTrash does not take a path - it instead returns files which are in any trashcan directory in the subsystem, sorted by parent directory.

UserIdP and RealmIdP are optional arguments which can be utilized by users with sufficient authority to read other user's trash. By default, if UserIdP is NULL the current authenticated user will be provided, and if RealmIdP is used the local realm will be used. If some other uid and realm are provided, trash will be read for that user.

Parameters

in	*SubsystemId*	directory object handle
in	*UserIdP*	user id - if NULL, the authenticated user's credentials are used
in	*RealmIdP*	realm id - if NULL, the local realm is used
in	*BufferSize*	size of output buffer (in bytes)
out	*End*	hit end of trash
out	*TrashPtr*	directory entry information

Return values

>=0	No error. Return value is the number of entries copied to the output buffer.
-ERANGE	The buffer size was not large enough to return entries
-EFAULT	One of the End, or TrashPtr parameters is NULL.
-EINVAL	BufferSize is zero or a RealmId was provided with no UserId.

Note

Trashcan paths are from their fileset root. For systems which make use of filesets this may not be an absolute path. To obtain an absolute path, the handle should be traversed back to the root of roots. See hpss_GetFullPath() for more information.

9.20.2.5 int hpss_Undelete (const char ∗ *Path,* hpss_undelete_flags_t *Flag*)

Restore a namespace entry from the trash.

The 'hpss_Undelete' function causes an object to be moved from its current position (most likely a trashcan) back to its position before it was deleted. This also causes the file to no longer be a candidate for trashcan cleanup. Optionally, the function can revert the timestamps (a la hpss_Utime) to their values prior to the deletion.

Parameters

in	*Path*	Path of file to undelete
in	*Flag*	Flags modifying the behavior of the undelete

Return values

0	Undelete was successful.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

Note

hpss_Undelete is provided as a convenience function, but its current implementation may not be robust enough for some applications. hpss_Undelete has some significant limitations which are described below: The way hpss_Undelete works is that it retrieves the trashcan info for the file, looks up the original path to the file, and renames the file back to its original position. If the flags are set it may overwrite an existing file and/or restore the timestamps. These operations are subject to potential data races with other clients who may be restoring other versions of the same file path. What hpss_Undelete does not do is re-create the existing path, or look up where its original parent may have been moved to; it merely attempts to go back to its original location, and it is up to the application as to whether that should be recreated or not, or perhaps recovered from the trash if it exists. Performing something like a recursive undelete of a directory is especially challenging due to the flat trashcan. Undeleting a directory from the trashcan will not restore its contents; they must be restored individually. This would generally require a scan of the trashcan, and/or recreating the required directory structures within the application processing.

9.20.2.6 int hpss_UndeleteHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* hpss_undelete_flags_t *Flag*)

Restore a namespace entry from the trash.

The 'hpss_UndeleteHandle' function causes an object to be moved from its current position (most likely a trashcan) back to its position before it was deleted. This also causes the file to no longer be a candidate for trashcan cleanup.

Parameters

in	*ObjHandle*	Parent object handle
in	*Path*	Path of file to undelete
in	*Ucred*	User credentials
in	*Flag*	Flags modifying the behavior of the undelete

Return values

0	Undelete was successful.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

Note

See hpss_Undelete() for a discussion of the limitations of this functionality before using.

9.21 HPSS Environment Variable Module

Prototypes for HPSS Environment variable functions.

Functions

- signed32 hpss_Envcpy (char ∗Buf, char const ∗Env, unsigned32 Bufsize)

  Copies an environment value into a user-supplied buffer.
- signed32 hpss_EnvLocalUpdate (char const ∗EnvStr)

  Update an environment table entry within the local process.
- void hpss_EnvReInitialize (void)

  Reinitialize the environment variables from the override file.
- char ∗ hpss_Getenv (char const ∗Env)

  Retrieve HPSS and system environment variables.

9.21.1 Detailed Description

Prototypes for HPSS Environment variable functions.

9.21.2 Function Documentation

9.21.2.1 signed32 hpss_Envcpy (char ∗ *Buf,* char const ∗ *Env,* unsigned32 *Bufsize*)

Copies an environment value into a user-supplied buffer.

Parameters

`in`	*Buf*	User supplied string buffer
`in`	*Env*	Name of the environment variable
`in`	*Bufsize*	Size of *Buf*

This function calls hpss_Getenv to get the environment value for the variable defined in Env. The caller should supply a string buffer large enough to contain the resulting string from hpss_Getenv. Bufsize is the length of the buffer in characters. If the string found in the environment will not fit in the buffer, hpss_Envcpy returns HPSS_ENAMETOOLONG, and the buffer will contain the truncated value.

Return values

HPSS_E_NOERROR	Success
HPSS_ENOMEM	*Buf* NULL or 0 Length
HPSS_ENAMETOOLONG	*Buf* too small
HPSS_ENOENT	*Env* is not defined

9.21.2.2 signed32 hpss_EnvLocalUpdate (char const ∗ *EnvStr*)

Update an environment table entry within the local process.

This function will write an HPSS environment override value to the override array.

Return values

HPSS_E_NOERROR	Success
HPSS_EAGAIN	Environment not initialized
HPSS_ENOMEM	Memory allocation error
HPSS_EINVAL	Invalid input argument

Note

This function changes only the environment local to the running process which called this function.

9.21.2.3 void hpss_EnvReInitialize (void)

Reinitialize the environment variables from the override file.

Reset the internal variables so this package will reconstruct its cached values from the env.conf file. This allows changes to the file to be picked up by this package without restarting the client

9.21.2.4 char∗ hpss_Getenv (char const ∗ *Env*)

Retrieve HPSS and system environment variables.

Parameters

`in`	*Env*	Name of environment variable

This function is a wrapper to getenv. It first checks to see if there is an HPSS default defined. If there is an HPSS default defined that is returned (after any expansion). If it is not defined then the storage from getenv is returned.

9.22 HPSS Networking Library

Functions for facilitating IPV4 and IPV6 Networking.

Functions

- int hpss_net_accept (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Takes a connection request from the queue and creates a new socket.
- int hpss_net_addrmatch (const hpss_sockaddr_t ∗Address, const hpss_sockaddr_t ∗Mask, const hpss_-sockaddr_t ∗Entry)

  Compare addresses using netmask.
- int hpss_net_bind (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Bind an unnamed socket to an address.
- int hpss_net_connect (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Forms a network connection between a socket and an address.
- void hpss_net_generatemask (const hpss_sockaddr_t ∗source, hpss_sockaddr_t ∗mask, int bits)

  Generate a netmask for a given number of significant bits.
- int hpss_net_get_family_option (char ∗errbuf, size_t errbuflen)

  Retrieves the HPSS_NET_FAMILY environment variable.
- char ∗ hpss_net_get_family_string (void)

  Get the environment variable HPSS_NET_FAMILY string.
- void ∗ hpss_net_getinaddr (const hpss_sockaddr_t ∗addr, unsigned int ∗addrfamily, socklen_t ∗addrlen)

  Return a pointer to the internal sockaddr structure for the protocol.
- int hpss_net_getnameinfo (const hpss_sockaddr_t ∗addr, char ∗host, size_t hostlen, char ∗serv, size_t servlen, int flags, char ∗errbuf, size_t errbuflen)

  Return host and service string for provided socket address.
- int hpss_net_getpeername (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Return a socket address for a remote peer.
- int hpss_net_getport (const hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Return the port associated with a socket address.
- int hpss_net_getuniversaladdress (const hpss_sockaddr_t ∗addr, char ∗buf, int buflen, char ∗errbuf, size_t errbuflen)

  Return the string representation of the socket address structure.
- void hpss_net_initaddr (hpss_sockaddr_t ∗addr)

  Initializes a hpss_sockaddr_t.
- int hpss_net_listen (int sockfd, int backlog, char ∗errbuf, size_t errbuflen)

  Allows a socket to be used to accept incoming connections.
- int hpss_net_setport (hpss_sockaddr_t ∗addr, unsigned short port, char ∗errbuf, size_t errbuflen)

  Update the port associated with a socket address.
- int hpss_net_socket (hpss_sockaddr_t ∗addr, int type, int protocol, char ∗errbuf, size_t errbuflen)

  Open a new socket descriptor.
- void hpss_net_strerror (int errnum, char ∗buf, size_t buflen)

  Fill in the buffer with error string.
- unsigned short hpss_net_universaladdresstoport (const char ∗address)

  Return the port from the universal address representation of the address structure.

9.22.1 Detailed Description

Functions for facilitating IPV4 and IPV6 Networking.

9.22.2 Function Documentation

9.22.2.1 int hpss_net_accept (int *sockfd,* hpss_sockaddr_t ∗ *addr,* char ∗ *errbuf,* size_t *errbuflen*)

Takes a connection request from the queue and creates a new socket.

Parameters

in	*sockfd*	Listening socket
out	*addr*	Sockaddr of the client
out	*errbuf*	Buffer for more detailed error info
in	*errbuflen*	Length of errbuf

hpss_net_accept() takes a socket, *sockfd*, that has been created and bound to an address with hpss_net_socket(). It will take a connection request from the queue of pending connections, and will create a new socket with the same properties of sockfd. This function behaves in a similar manner to accept(2).

Return values

Positive	File descriptor of the accepted client socket
-1	Error; refer to errno for specific error code

9.22.2.2 int hpss_net_addrmatch (const hpss_sockaddr_t ∗ *Address,* const hpss_sockaddr_t ∗ *Mask,* const hpss_sockaddr_t ∗ *Entry*)

Compare addresses using netmask.

Parameters

in	*Address*	First network address
in	*Mask*	Network mask
in	*Entry*	Second network address

Compares two network addresses, byte by byte. If *Mask* is NULL, then all address bits are significant.

Return values

0	No matching entry
1	Addresses match

9.22.2.3 int hpss_net_bind (int *sockfd,* hpss_sockaddr_t ∗ *addr,* char ∗ *errbuf,* size_t *errbuflen*)

Bind an unnamed socket to an address.

Parameters

in	*sockfd*	Listening socket
out	*addr*	Sockaddr of the client
out	*errbuf*	Error buffer
in	*errbuflen*	Length of the error buffer

hpss_net_bind() binds an unnamed socket specified by *sockfd* to the address specified by *addr*. This function behaves in a similar manner to bind(2).

Return values

0	Success
-1	Error detected; errno set to appropriate value

Note

This function may change the scope id (sin6_scope_id) for the hpss_sockaddr_t, addr, structure. Currently this is only done on Linux systems.

9.22.2.4 int hpss_net_connect (int *sockfd*, hpss_sockaddr_t ∗ *addr*, char ∗ *errbuf*, size_t *errbuflen*)

Forms a network connection between a socket and an address.

Parameters

`in`	*sockfd*	Socket to be connected
`in,out`	*addr*	Sockaddr to connect to
`out`	*errbuf*	Buffer for more detailed error info
`in`	*errbuflen*	Length of the error buffer

hpss_net_connect() takes a socket, *sockfd*, and an hpss_sockaddr_t, *addr*, and forms a network connection between them. This function behaves in a similar manner to connect(2).

Return values

0	Success
-1	Error detected; errno set appropriately

Note

This function may change the scope id (sin6_scope_id) for the hpss_sockaddr_t, addr, structure. Currently this is only done on Linux systems.

9.22.2.5 void hpss_net_generatemask (const hpss_sockaddr_t ∗ *source,* hpss_sockaddr_t ∗ *mask,* int *bits*)

Generate a netmask for a given number of significant bits.

Parameters

`in`	*source*	Template address
`out`	*mask*	Generated netmask
`in`	*bits*	Number of significant bits

This function generates a netmask based on the number of significant bits desired. A 0 can be passed in, which means that all bits are significant.

9.22.2.6 int hpss_net_get_family_option (char ∗ *errbuf,* size_t *errbuflen*)

Retrieves the HPSS_NET_FAMILY environment variable.

Parameters

`out`	*errbuf*	Buffer for more detailed error info
`in`	*errbuflen*	Length of errbuf

hpss_net_get_family_option() is used to get the environment variable HPSS_NET_FAMILY. This variable should be set to one of: "ipv4_only", "ipv6", or "ipv6_only". It is used to make decisions about how the HPSS site wants to run.

Outputs: This function returns an integer corresponding to the address family desired. If HPSS_NET_FAMILY is set to an invalid value, the function returns the error HPSS_NET_UNSPECIFIED and fills in errbuf if it is passed in.

Return values

HPSS_NET_IPV6	IPV6 / IPV4 Mixed Mode
HPSS_NET_IPV6_ONLY	IPV6 Only
HPSS_NET_IPV4_ONLY	IPV4 Only
HPSS_NET_UNSPECIFIED	Unspecified

9.22.2.7 char∗ hpss_net_get_family_string (void)

Get the environment variable HPSS_NET_FAMILY string.

hpss_net_get_family_string() is used to get the environment variable HPSS_NET_FAMILY. It exists so that we properly use hpss_Getenv() or getenv() depending on the compiler directive HPSS_EXT_CLIENT.

Returns

The value of the HPSS_NET_FAMILY environment variable

Note

Compiling with HPSS_EXT_CLIENT allows an external client to specify this setting without requiring the hpss-_Getenv library.

9.22.2.8 void∗ hpss_net_getinaddr (const hpss_sockaddr_t ∗ *addr,* unsigned int ∗ *addrfamily,* socklen_t ∗ *addrlen*)

Return a pointer to the internal sockaddr structure for the protocol.

Parameters

`in`	*addr*	Socket address to get the internal representation of the address from
`out`	*addrfamily*	Address family for addr (AF_INET or AF_INET6)
`out`	*addrlen*	Address length for addr

hpss_net_getinaddr() will return the internal sin6_addr for IPv6 addresses or sin_addr for IPv4 addresses.

Returns

A pointer to the struct sockaddr_in6 structure if the hpss_sockaddr_t contains an IPv6 address; a struct sockaddr_in if the hpss_sockaddr_t contains an IPv4 address; otherwise NULL is returned. If addrfamily and/or addrlen are not NULL then return the address family and the address length as well.

9.22.2.9 int hpss_net_getnameinfo (const hpss_sockaddr_t ∗ *addr,* char ∗ *host,* size_t *hostlen,* char ∗ *serv,* size_t *servlen,* int *flags,* char ∗ *errbuf,* size_t *errbuflen*)

Return host and service string for provided socket address.

Parameters

`in`	*addr*	Socket address
`out`	*host*	Hostname
`in`	*hostlen*	Size of the host buffer
`out`	*serv*	Service/Port
`in`	*servlen*	Size of the service buffer
`in`	*flags*	Flags to send getnameinfo
`out`	*errbuf*	Buffer for detailed error info
`in`	*errbuflen*	Length of errbuf

hpss_net_getnameinfo() behaves exactly like getnameinfo(3), but instead with a hpss_sockaddr_t. It takes the hpss_sockaddr_t,a passed in and returns a hostname string and a service string.

Return values

0	Success
EAI_SYSTEM	Check errno for more details

Other	EAI error code

Note

getnameinfo() returns non-POSIX error codes in general. In order to decode the error gai_strerror() should be used.

9.22.2.10 int hpss_net_getpeername (int *sockfd*, hpss_sockaddr_t ∗ *addr*, char ∗ *errbuf*, size_t *errbuflen*)

Return a socket address for a remote peer.

Parameters

`in`	*sockfd*	Socket to get the remote peer from
`out`	*addr*	Socket address structure that will contain peer information
`out`	*errbuf*	Buffer for more detailed error info
`in`	*errbuflen*	Length of errbuf

hpss_net_getpeername() behaves exactly like getpeername(2), but instead with a hpss_sockaddr_t. It takes the socket descriptor, passed in and returns a socket address structure that represents the remote peer.

Return values

0	Success
-1	Errno set to POSIX error code

9.22.2.11 int hpss_net_getport (const hpss_sockaddr_t ∗ *addr*, char ∗ *errbuf*, size_t *errbuflen*)

Return the port associated with a socket address.

Parameters

`in`	*addr*	Socket address to get port from
`out`	*errbuf*	Buffer for detailed error info
`in`	*errbuflen*	Length of errbuf

hpss_net_getport() is a convenience function that returns the port that is associated with the socket address provided.

Return values

0-USHRT_MAX	Success (port number)
$<$*0*	Negated POSIX error code

9.22.2.12 int hpss_net_getuniversaladdress (const hpss_sockaddr_t ∗ *addr*, char ∗ *buf*, int *buflen*, char ∗ *errbuf*, size_t *errbuflen*)

Return the string representation of the socket address structure.

Parameters

`in`	*addr*	Socket address structure to get the universal address from
`out`	*buf*	Buffer to store the universal address in

in	*buflen*	Length of buf
out	*errbuf*	Buf for detailed error info
in	*errbuflen*	Length of errbuf

hpss_net_getuniversaladdress() returns the string representation of the socket address structure in the form: [ipaddress].[port_high].[port_low] This is used as a protocol independent representation of the socket address when talking to the rpcbind daemon.

Return values

>0	Length written to buf
<=0	Negated POSIX error code

9.22.2.13 void hpss_net_initaddr (hpss_sockaddr_t * *addr*)

Initializes a hpss_sockaddr_t.

Parameters

out	*addr*	Socket address to initialize

hpss_net_initaddr() initializes a hpss_sockaddr_t. It will set addr->hs_addrlen to sizeof(struct sockaddr_storage) and zero out addr->hs_addr.

9.22.2.14 int hpss_net_listen (int *sockfd*, int *backlog*, char * *errbuf*, size_t *errbuflen*)

Allows a socket to be used to accept incoming connections.

Parameters

in	*sockfd*	Socket to perform the listen on
in	*backlog*	Size of the request queue
out	*errbuf*	Buffer for more detailed error info
in	*errbuflen*	Length of errbuf

hpss_net_listen() behaves exactly like listen(2). It takes the socket descriptor, and a queue size and allows the socket to be used to accept incoming connections.

Return values

0	Success
-1	Errno set appropriately

9.22.2.15 int hpss_net_setport (hpss_sockaddr_t * *addr*, unsigned short *port*, char * *errbuf*, size_t *errbuflen*)

Update the port associated with a socket address.

Parameters

in	*addr*	Sockaddr to store port
in	*port*	Port to store
out	*errbuf*	Buffer for more detailed error info
in	*errbuflen*	Length of errbuf

hpss_net_setport() is a convenience function that can be used to update the port that is associated with the socket address.

Return values

0	Success
Other	Negated POSIX error code

9.22.2.16 int hpss_net_socket (hpss_sockaddr_t ∗ *addr,* int *type,* int *protocol,* char ∗ *errbuf,* size_t *errbuflen*)

Open a new socket descriptor.

Parameters

`in`	*addr*	Sockaddr to get family from
`in`	*type*	Type of socket (TCP, UDP, etc.)
`in`	*protocol*	Protocol to use with socket
`out`	*errbuf*	Buf for detailed error info
`in`	*errbuflen*	Length of errbuf

hpss_net_socket() behaves similarly to socket(2). The addr parameter should be an hpss_sockaddr_t that was previously set up with hpss_net_getaddrinfo(). The type and protocol parameters are the same as what normally would be provided to socket(2).

Return values

>=0	Valid file descriptor
-1	Errno set appropriately

9.22.2.17 void hpss_net_strerror (int *errnum,* char ∗ *buf,* size_t *buflen*)

Fill in the buffer with error string.

Parameters

`in`	*errnum*	Error code to use
`out`	*buf*	Buffer to be returned
`in`	*buflen*	Size of the buffer

hpss_net_strerror() fills in the buffer, buf, with a string that describes the error indicated by errnum.

Note

This function was written to behave identically on several different platforms whether the XSI-compliant version or the GNU version of strerror_r() is getting used. Not that it will save and restore errno in case the XSI-compliant version of strerror_r() is used.

9.22.2.18 unsigned short hpss_net_universaladdresstoport (const char ∗ *address*)

Return the port from the universal address representation of the address structure.

Parameters

`in`	*address*	universal address representation string

hpss_net_universaladdresstoport() returns the port from the universal address representation of the address structure: [ipaddress].[port_high].[port_low] This is used as a protocol independent representation of the socket address when talking to the rpcbind daemon.

Return values

$>$*0*	Port Number
0	Could not retrieve the port number

9.23 HPSS XML Interface

XML Interfaces.

Functions

- char ∗ hpss_ChompXMLHeader (char ∗XML, char ∗Header)

  Remove the XML Header from an XML string, and optionally return it in its own string.
- int hpss_CreateXMLWhere (hpss_userattr_list_t ∗Attrs, char ∗Query, int QueryLen)

  Generates XMLExists statements.
- int hpss_CreateXQueryDelete (char ∗XPath, char ∗Query, int QueryLen)

  Generates XQuery Delete statements.
- int hpss_CreateXQueryGet (char ∗XPath, int XMLFlags, char ∗Query, int QueryLen)

  Generates XQuery Get statements.
- int hpss_CreateXQueryUpdate (char ∗XPath, char ∗Value, char ∗Query, int QueryLen)

  Generate an insert or replace command from an XPath, value, and schema name.
- int hpss_ValidXPath (char ∗XPath)

  Validates that the argument has some of the required components of an XPath statement.
- int hpss_XMLtoAttr (char ∗XMLstr, hpss_userattr_list_t ∗Attrs, int Flags)

  Convert an XML string into some number of Key/Value pairs.
- int hpss_XPathBranch (char ∗Path, char ∗Ret)

  Obtain the XPath without the leaf component.
- int hpss_XPathLeaf (char ∗Path, char ∗Ret)

  Obtain the last component of an XPath.
- int hpss_XPathRBranch (char ∗Path, char ∗Ret)

  Obtain the XPath without the root component.
- int hpss_XPathRoot (char ∗Path, char ∗Ret)

  Obtain the first component of an XPath.
- int hpss_XPathtoXML (char ∗Attr, char ∗Val, char ∗XMLstr, int Len, int Insert, int Level)

  Convert an attribute/value pair from XPath to an XML string.

9.23.1 Detailed Description

XML Interfaces.

9.23.2 Function Documentation

9.23.2.1 char∗ hpss_ChompXMLHeader (char ∗ *XML*, char ∗ *Header*)

Remove the XML Header from an XML string, and optionally return it in its own string.

Parameters

`in`	*XML*	XML String
`in,out`	*Header*	Header component of XML

Return values

NULL	No header to remove, or resources unavailable
Other	Success

Note

The caller must free the returned string. Algorithm:

- Find the beginning of the header.
- Find the end of the header.
- If requested, copy the header to the Header variable.
- Copy everything but the header over the orignal string.

9.23.2.2 int hpss_CreateXMLWhere (hpss_userattr_list_t ∗ *Attrs,* char ∗ *Query,* int *QueryLen*)

Generates XMLExists statements.

Parameters

`in`	*Attrs*	Attribute List (only the first element is used)
`out`	*Query*	Query Buffer
`in`	*QueryLen*	Query Length

Return values

0	Success
-EINVAL	String exceeds supplied length

Note

Algorithm:

- Generate the XMLExists depending upon whether the Attr has a value

9.23.2.3 int hpss_CreateXQueryDelete (char ∗ *XPath,* char ∗ *Query,* int *QueryLen*)

Generates XQuery Delete statements.

Parameters

`in`	*XPath*	XPath to Generate Delete Query For
`out`	*Query*	Query Buffer
`in`	*QueryLen*	Length of Query Buffer

Return values

0	Success
-EINVAL	String exceeds supplied length

Note

Algorithm:

- If needed, generate an XMLVALIDATE
- Generate xmlquery preamble
- Generate delete statement
- Generate xmlquery close

9.23.2.4 int hpss_CreateXQueryGet (char ∗ *XPath,* int *XMLFlags,* char ∗ *Query,* int *QueryLen*)

Generates XQuery Get statements.

Parameters

in	*XPath*	XPath to Generate Retrieval Query For
in	*XMLFlags*	Flags for Retrieval Processing • UDA_API_XML retrieves raw XML • UDA_API_VALUE retrieves XML attribute value
out	*Query*	Query Buffer
in	*QueryLen*	Query Buffer Length

Return values

0	Success
-EINVAL	String exceeds supplied length

Note

Algorithm:

- If needed, generate an XMLVALIDATE
- Generate xmlquery preamble
- Generate delete statement
- Generate xmlquery close

9.23.2.5 int hpss_CreateXQueryUpdate (char ∗ *XPath,* char ∗ *Value,* char ∗ *Query,* int *QueryLen*)

Generate an insert or replace command from an XPath, value, and schema name.

Generate an XQuery Insert or replace command given the XPath, Value, and Schema. This function decomposes the XPath and converts it into XML in order to generate statements covering all cases of tag availability.

For example, given /hpss/some/test, cases will be generated for replacing if /hpss/some/test exists, inserting

```
<test>..</test>
```

if only /hpss/some exists, or inserting

```
<some><test>...</test></some>
```

if only /hpss exists.

Parameters

in	*XPath*	XPath to Generate Update Query For
in	*Value*	Value to set in Update Query
out	*Query*	Buffer for Generated Query
in	*QueryLen*	Length of Query Buffer

Return values

0	Success
-EINVAL	String exceeds supplied length
-ENOMEM	Ran out of memory
Others	Error converting XPath to XML

Note

Algorithm:

- Create XUpdate xmlquery preamble
- Create XUpdate replace
- For each additional layer of tag depth, generate a insert statement corresponding to that layer.
- Close XUpdate xmlquery statement

9.23.2.6 int hpss_ValidXPath (char ∗ *XPath*)

Validates that the argument has some of the required components of an XPath statement.

Parameters

`in`	*XPath*	XPath String to Verify

Return values

0	Valid
-EFAULT	No XPath supplied
-EINVAL	Invalid XPath

Note

This function does bare bones checking required by other functions, such as the Leaf/Root/Branch functions. Algorithm:

- Check that the string is valid.
- Check that the string begins with a /, for the root node.
- Check that the string does not end with a /.

9.23.2.7 int hpss_XMLtoAttr (char ∗ *XMLstr,* hpss_userattr_list_t ∗ *Attrs,* int *Flags*)

Convert an XML string into some number of Key/Value pairs.

Parameters

`in`	*XMLstr*	XML String to process
`out`	*Attrs*	User-defined Attributes List from XML
`in`	*Flags*	Flags to change processing behavior • XML_NO_ATTR will only handle XML Element Nodes • XML_ATTR will also handle XML Element Attributes

Return values

0	Success
-EFAULT	XMLstr or Attrs is NULL
-EINVAL	Invalid flag specified or invalid XML string
-ENOENT	Number of attributes collected did not match number of attributes found
-ENOMEM	Out of memory

Note

Uses state transitions to enforce XML syntax rules Uses an N-ary tree to hold the XML for post-processing This is required for multiple occurrence tracking

```
                         -----|
                         |    |
                         v    |
              |-------------| |
 Start------->|  XML_OPEN   |-|
   |          |-------------|
   |  ^ ^   |
   | / |   |
   v        / |   v
|-----------|       |------------|
| XML_CLOSE |<------|  XML_VALUE |
|-----------|       |------------|
   |   |  ^
   |   |  |
   v   ----
  End
```

9.23.2.8 int hpss_XPathBranch (char ∗ *Path,* char ∗ *Ret*)

Obtain the XPath without the leaf component.

Parameters

`in`	*Path*	Full XPath
`in,out`	*Ret*	All but last component of Path

Return values

-EFAULT	An argument was NULL
-EINVAL	The supplied XPath is invalid
-ENOENT	There is no branch component
0	Success

Note

Algorithm:

- If Path or Ret is invalid Return error
- Find last occurance of '/' Copy from first '/' to last '/'
- If not found return ENOENT

9.23.2.9 int hpss_XPathLeaf (char ∗ *Path,* char ∗ *Ret*)

Obtain the last component of an XPath.

Parameters

`in`	*Path*	Full XPath
`in,out`	*Ret*	Last component of Path

Return values

-EFAULT	An argument was NULL
-EINVAL	The supplied XPath is invalid
-ENOENT	There is no leaf component
-ERANGE	XPath component is too large
0	Success

Note

Algorithm:

- If Path or Ret is invalid Return error
- Find last occurance of '/' Copy from last '/' to end
- If not found return ENOENT

9.23.2.10 int hpss_XPathRBranch (char ∗ *Path,* char ∗ *Ret*)

Obtain the XPath without the root component.

Parameters

`in`	*Path*	Full XPath
`in,out`	*Ret*	All but first component of Path

Return values

-EFAULT	An argument was NULL
-EINVAL	The supplied XPath is invalid
-ENOENT	There is no RBranch component
0	Success

Note

Algorithm:

- If Path or Ret is invalid Return error
- Find second occurance of '/' Copy from second '/' to last '/'
- If not found return ENOENT

9.23.2.11 int hpss_XPathRoot (char ∗ *Path,* char ∗ *Ret*)

Obtain the first component of an XPath.

Parameters

`in`	*Path*	Full XPath

in,out	*Ret*	First Component of Path

Return values

-EFAULT	An argument was NULL
-EINVAL	The supplied XPath is invalid
-ENOENT	There is no first component
0	Success

Note

Algorithm:

- If Path or Ret is invalid Return error
- Find second occurance of '/' Copy from first '/' to second
- If not found return ENOENT

9.23.2.12 int hpss_XPathtoXML (char ∗ *Attr,* char ∗ *Val,* char ∗ *XMLstr,* int *Len,* int *Insert,* int *Level*)

Convert an attribute/value pair from XPath to an XML string.

Parameters

in	*Attr*	Attribute Name
in	*Val*	Attribute Value
in,out	*XMLstr*	XML String
in	*Len*	Length of XML String
in	*Insert*	Flag for modifying value expression when used for insertion.
in	*Level*	Level of XML composition to perform • A level of 2 would create XML that was two levels deep, but no deeper • MAX_DEPTH is the highest legal value for Level

Return values

0	Success
-EINVAL	Length supplied is invalid; less than 0 or greater than MAX_DEPTH
-EINVAL	Attr or Value contained the NULL string
-EINVAL	XMLStr is too short
-EFAULT	One or more required arguments were NULL
-ENOMEM	Ran out of memory

Note

Algorithm:

Decompose the XPath into individual leaf components leading back to the root. eg. /hpss/test/attr –> [attr, test, hpss] Write each leaf (starting from the last one) to the string as an open XML tag eg. = [attr, test, hpss] –>

```
<hpss><test><attr>
```

Write the value to the string Note: We may write the value in a way suitable for detecting and handling whitespace in an XQUERY insert if required. eg.

```
<hpss><test><attr>value
```

or eg.

```
<hpss><test><attr>{xs:string("value")}
```

(for XQUERY) Write each leaf (starting with the first one) to the string as a closing XML tag eg. [attr, test, hpss] –>

```
<hpss><test><attr>value</attr></test></hpss>
```

Clean up leaf components

9.24 64-bit Math Library

HPSS 64-bit Math Library.

Macros

- #define add64_3m(a, b, c)
- #define add64m(a1, a2)
- #define and64m(a1, a2)
- #define andbit64m(a, b)
- #define bld64m(a, b)
- #define cast32m(a)
- #define cast64m(a)
- #define chkbit64m(a, b)
- #define clrbit64m(a, b)
- #define CONVERT_LONGLONG_TO_U64(LL, U64)
- #define CONVERT_U64_TO_LONGLONG(U64, LL)
- #define dec64m(a1, a2)
- #define div2x64m(a1, a2)
- #define div64_3m(a, b, c)
- #define div64m(a1, a2)
- #define eq64m(a1, a2)
- #define eqz64m(a)
- #define ge64m(a1, a2)
- #define gt64m(a1, a2)
- #define high32m(a)
- #define inc64m(a1, a2)
- #define le64m(a1, a2)
- #define low32m(a)
- #define lt64m(a1, a2)
- #define mem64m(a)
- #define mod2x64m(a1, a2)
- #define mod64_3m(a, b, c)
- #define mod64m(a1, a2)
- #define mul64_3m(a, b, c)
- #define mul64m(a1, a2)
- #define neq64m(a1, a2)
- #define neqz64m(a)
- #define not64m(a)
- #define or64m(a1, a2)
- #define orbit64m(a, b)
- #define shl64_3m(a, b, s)
- #define shl64_ipm(a, s)
- #define shl64m(a1, a2)
- #define shr64_3m(a, b, s)
- #define shr64_ipm(a, s)
- #define shr64m(a1, a2)
- #define sub64_3m(a, b, c)
- #define sub64m(a1, a2)

9.24.1 Detailed Description

HPSS 64-bit Math Library.

Deprecated

It is the intent that u_signed64 and the manipulation macros will be removed from the HPSS Client API in the next major version (version 8). Application developers should prepare to use standard 64-bit C types referenced in stdint.h in order to make porting to version 8 easier.

9.24.2 Macro Definition Documentation

9.24.2.1 #define add64_3m(*a,* *b,* *c*)

Assign *a* to *b* + *c*

This function should not be used with a non-literal as a as it will be evaluated multiple times.

9.24.2.2 #define add64m(*a1,* *a2*)

Add two unsigned 64s, *a1* and *a2*

9.24.2.3 #define and64m(*a1,* *a2*)

Find the bitwise and of two 64-bit values, *a1* and *a2*

9.24.2.4 #define andbit64m(*a,* *b*)

And bit position *b* in unsiqned 64 bit integer *a*

9.24.2.5 #define bld64m(*a,* *b*)

Build a unsigned64 from 32-bit integers *a* and *b*

9.24.2.6 #define cast32m(*a*)

Cast *a* to an unsigned 32 bit integer

9.24.2.7 #define cast64m(*a*)

Cast *to* a unsigned 64

9.24.2.8 #define chkbit64m(*a,* *b*)

Return 1 if bit position *b* is set in unsigned integer *a*

9.24.2.9 #define clrbit64m(*a*, *b*)

Clear bit position *b* in unsigned 64 bit integer *a*

9.24.2.10 #define CONVERT_LONGLONG_TO_U64(*LL*, *U64*)

Convert a long long, *LL* to a u_signed64, *U64*

9.24.2.11 #define CONVERT_U64_TO_LONGLONG(*U64*, *LL*)

Convert a u_signed64, *U64* to a long long, *LL*

9.24.2.12 #define dec64m(*a1*, *a2*)

Subtract *a2* from *a1* destroying old *a1*.

9.24.2.13 #define div2x64m(*a1*, *a2*)

Divide two unsigned 64s, *a1* and *a2*

9.24.2.14 #define div64_3m(*a*, *b*, *c*)

Assign *a* to *b* divided by *c*

This macro is fast if using 32-bit arithmetic.

This macro should not be used with a macro as *a* as it will be evaluated multiple times.

9.24.2.15 #define div64m(*a1*, *a2*)

Divide a unsigned 64, *a1*, by a unsigned 32 *a2*

9.24.2.16 #define eq64m(*a1*, *a2*)

Returns TRUE if *a1* is equal to *a2*, or FALSE.

9.24.2.17 #define eqz64m(*a*)

Returns TRUE if *a* equals zero, or FALSE.

9.24.2.18 #define ge64m(*a1*, *a2*)

Returns TRUE if *a1* is greater than or equal to *a2*, or FALSE.

9.24.2.19 #define gt64m(*a1,* *a2*)

Returns TRUE if *a1* is greater than *a2*, or FALSE.

9.24.2.20 #define high32m(*a*)

Reference the upper 32 bits of unsigned64 *a*

9.24.2.21 #define inc64m(*a1,* *a2*)

Add *a2* to *a1* destroying old *a1*.

9.24.2.22 #define le64m(*a1,* *a2*)

Returns TRUE if *a1* is less than or equal to *a2*, or FALSE.

9.24.2.23 #define low32m(*a*)

Reference the lower 32 bits of unsigned64 *a*

9.24.2.24 #define lt64m(*a1,* *a2*)

Returns TRUE if *a1* is less than *a2*, or FALSE.

9.24.2.25 #define mem64m(*a*)

Build a 64-bit unsigned int from *a*, an array of unsigned characters

9.24.2.26 #define mod2x64m(*a1,* *a2*)

Modulus an unsigned 64, *a1* by an unsigned 64 *a2*

9.24.2.27 #define mod64_3m(*a,* *b,* *c*)

Assign *a* to *b* modulus *c*

This macro is fast if using 32-bit arithmetic.

This macro should not be used with a macro as *a* as it will be evaluated multiple times.

9.24.2.28 #define mod64m(*a1,* *a2*)

Modulus an unsigned 64, *a1* by an unsigned 32 *a2*

9.24.2.29 #define mul64_3m(*a*, *b*, *c*)

Assign *a* to *b* multiplied by *c*

This macro is fast if using 32-bit arithmetic.

This macro should not be used with a macro as *a* as it will be evaluated multiple times.

9.24.2.30 #define mul64m(*a1*, *a2*)

Multiply an unsigned 64, *a1* by an unsigned 32 *a2*

9.24.2.31 #define neq64m(*a1*, *a2*)

Returns TRUE if *a1* is not equal to *a2*, or FALSE.

9.24.2.32 #define neqz64m(*a*)

Returns TRUE if *a* does not equal zero, or FALSE.

9.24.2.33 #define not64m(*a*)

Find the bitwise not of a 64-bit value

9.24.2.34 #define or64m(*a1*, *a2*)

Find the bitwise or of two 64-bit values, *a1* and *a2*

9.24.2.35 #define orbit64m(*a*, *b*)

Or bit position *b* in unsigned 64 bit integer *a*

9.24.2.36 #define shl64_3m(*a*, *b*, *s*)

Assign *a* to *b* shifted *s* bytes left

This function should not be used with a non-literal as a as it will be evaluated multiple times.

9.24.2.37 #define shl64_ipm(*a*, *s*)

Shift *a* left by *s* positions, destroying old *a*.

9.24.2.38 #define shl64m(*a1*, *a2*)

Shift an unsigned 64, *a1* left by an unsigned 32 *a2*

9.24.2.39 #define shr64_3m(*a,* *b,* *s*)

Assign *a* to *b* shifted *s* bytes right

This function should not be used with a non-literal as a as it will be evaluated multiple times.

9.24.2.40 #define shr64_ipm(*a,* *s*)

Shift *a* right by *s* positions, destroying old *a*.

9.24.2.41 #define shr64m(*a1,* *a2*)

Shift an unsigned 64, *a1* right by an unsigned 32 *a2*

9.24.2.42 #define sub64_3m(*a,* *b,* *c*)

Assign *a* to *b* - *c*

This function should not be used with a non-literal as a as it will be evaluated multiple times.

9.24.2.43 #define sub64m(*a1,* *a2*)

Subtract two unsigned 64s, *a1* and *a2*

9.25 HPSS Client API Extensions

Common header for HPSS Client API extensions.

Macros

- #define MIN_RESTRICTED_PORT 1024

  Get the range of restricted ports that are allowed.

Functions

- int hpsscfgx_ConfGetClientInterfaces (const char ∗hostName, const char ∗serverName, int ∗nwIfCount, hpss_sockaddr_t ∗∗nwIfList, char ∗∗∗nwIfNames)

  Lookup client interfaces by host.
- hpss_cfg_stanza_t ∗ hpsscfgx_ConfParse (char cfgFile[ ])

  Parse the configuration file.
- int hpsscfgx_ConfSetDirPaths (char ∗theDirString)

  Set the Directory Path(s) for the Configuration File.
- int hpsscfgx_ConfSetFileName (char ∗theFilename)

  Set the name of the config file.
- hpss_cfg_stanza_t ∗ hpsscfgx_LookupHostName (hpss_cfg_stanza_t ∗hostStanzaList, const char ∗hostName, int ignoreDomain)

  Find a substanza for a hostname.
- int hpsscfgx_NetoptFindEntry (hpss_sockaddr_t ∗NetAddr, netopt_entry_t ∗∗RetEntryPtr)

  Searches through the network option table, if present to find network options configured for the specified address.
- ssize_t hpsscfgx_NetoptGetWriteSize (int SocketDescriptor, hpss_sockaddr_t ∗IpAddr)

  Return the network (TCP/IP) write size to be used for the specified connection.
- int hpsscfgx_NetoptSetSock (int Fd, hpss_sockaddr_t ∗NetAddress)

  Adjust the Socket characteristics.
- int hpsscfgx_PatternMatch (char ∗theString, char ∗thePattern, int ∗patternError)

  Pattern match function.

9.25.1 Detailed Description

Common header for HPSS Client API extensions.

9.25.2 Macro Definition Documentation

9.25.2.1 #define MIN_RESTRICTED_PORT 1024

Get the range of restricted ports that are allowed.

Parameters

out	*minPort*	If non-NULL, set on exit to minimum restricted port
out	*maxPort*	If non-NULL, set on exit to maximum restricted port

This function is called to get the range of restricted ports that are allowed, based upon the environment variables: RPC_RESTRICTED_PORTS HPSS_PORT_RANGE

HPSS_PORT_RANGE takes precedence, if both are specified. This is for compatibility with PFTP.

If no ports are specified, or there is a specification error, the function returns: minPort = MIN_RESTRICTED_PORT maxPort = MAX_RESTRICTED_PORT

9.25.3 Function Documentation

9.25.3.1 int hpsscfgx_ConfGetClientInterfaces (const char ∗ *hostName,* const char ∗ *serverName,* int ∗ *nwIfCount,* hpss_sockaddr_t ∗∗ *nwIfList,* char ∗∗∗ *nwIfNames*)

Lookup client interfaces by host.

Parameters

in	*hostName*	NULL or client hostname to search for, if NULL then the local hostname is used. The default client is used if neither the local host entry nor the specified client hostname can be found.
in	*serverName*	NULL or server host to search for. If NULL, then the Default server host stanza is used, if found.
out	*nwIfCount*	Set on exit to count of interfaces
out	*nwIfList*	If non-NULL, set on exit to malloc'd list of local network addresses
out	*nwIfNames*	If non-NULL, set on exit to malllc'd table of local interface names

This function looks up a list of client interfaces that match the supplied hostname (if non-NULL).

The PFTP Client Interfaces section of the HPSS.conf file has the following format:

```
PFTP Client Interfaces = {
   Client_Hostname1 [Client_Hostname1Alias ...] = {
      Server_Host1 [Server_Host1Alias ...] = {
          xxx.xxx.xxx.xxx [:xxx.xxx.xxx.xxx ...] # colon-separated
                        list of IP addresses
          xxx.xxx.xxx.xxx # may have multiple lines with
                        colon-separated IP addresses

            OR
          nic_name1[:nic_name2...]   # colon-separated list of
                        local NIC names

            OR
          attr = {
            mtu = value   # specifies selection of NICs with MTU = value
            mtu = +value  # specifies selection of NICs with MTU >= value
            mtu = -value  # specifies selection of NICs with MTU <= value
            mtu = mtu_min-mtu_max  # specifies selection of NICs
                          with MTU in the range
                          mtu_min <= value <= mtu_max
          }  # end of attribute substanza
      }

      Server_Host1 [Server_Host1Alias ...] = {
      }

      Default  = {
      }
   } # end of Client_Hostname1 section

   Client_Hostname2 [Client_Hostname2Alias ...] = {
   } # end of Client_Hostname2 section
   .
   .
   .
   Default = {    # Default Client Host
   }
} # end of PFTP Client Interfaces section
```

All strings (client hostnames, server hostnames, NIC names) may use wildcard characters to allow for pattern-matching. The backslash character "\" may be used to prefix any character (e.g., "\:") that is actually part of the name.

Attributes such as MTU can only be applied to NICs on the local host, and are ignored if hostName refers to a different client host.

Return values

0	Success
HPSS_ENOENT	Hostname or server not found
-1	Other error detected

Note

Network interface attributes (e.g. MTU size) are only matchable if the client host (hostName) refers to the local host.

9.25.3.2 hpss_cfg_stanza_t* hpsscfgx_ConfParse (char *cfgFile[]*)

Parse the configuration file.

Parameters

`out`	*cfgFile*	Returned config file path name

This function is called to parse the configuration file, if it can be found, creating the internal "hpss_CfgEntries" table.

The Configuration File is looked for in a number of places, determined using the following algorithm:

1. The path(s) specified by calling hpsscfgx_ConfSetDirPaths().

2. The path(s) specified by the HPSS_CFG_FILE_PATH environment variable.

3. /usr/local/etc

4. /var/hpss/etc

The name of the Configuration File (Appended to the Path) is HPSS.conf unless the hpss_ConfSetFileName() call is explicitly performed prior to the call to hpsscfgx_ConfParse.

If the hpss_CfgEntries table has already been built, then this call does nothing. It is necessary to free and NULL hpss_CfgEntries to perform this routine more than one time.

Returns

Function returns a pointer to the parsed configuration stanza table on successful exit, or a NULL pointer on failure.

9.25.3.3 int hpsscfgx_ConfSetDirPaths (char * *theDirString*)

Set the Directory Path(s) for the Configuration File.

Parameters

`in`	*theDirString*	The directories to be used; multiples separated by a colon

This function provides an API to allow the application to supply an alternate Directory structure(s) for locating the configuration file This path(s)will be checked when the path/file is parsed (in function hpsscfgx_ConfParse).

Note

The following global variables are used as a result of this function call: hpss_ConfDirPaths [IN,OUT] contains copy of the passed in filename on successful exit.

Return values

HPSS_E_NOERROR	Success
HPSS_EFAULT	NULL string supplied
HPSS_ENOMEM	String cannot be copied to local storage

Note

1. If multiple calls are made, the last one overrides. 2. If this function is called after HPSS.conf has already been parsed, the call has no effect.

9.25.3.4 int hpsscfgx_ConfSetFileName (char ∗ *theFilename*)

Set the name of the config file.

Parameters

`in`	*theFilename*	The name of the configuration file

This function provides an API to allow the application to supply an alternate filename for the HPSS.conf file. This filename will be checked first when the path/file is parsed (in function hpsscfgx_ConfParse).

Note

The following global variables are used as a result of this function call: hpss_ConfFileName [IN,OUT] contains copy of the passed in filename on successful exit.

Return values

HPSS_E_NOERROR	Success
HPSS_EFAULT	NULL Filename supplied
HPSS_ENOMEM	Pathname cannot be copied to local storage

Note

1. If multiple calls are made, the last one overrides. 2. If this function is called after HPSS.conf has already been parsed, the call has no effect.

9.25.3.5 hpss_cfg_stanza_t∗ hpsscfgx_LookupHostName (hpss_cfg_stanza_t ∗ *hostStanzaList,* const char ∗ *hostName,* int *ignoreDomain*)

Find a substanza for a hostname.

Parameters

`in`	*hostStanzaList*	Head of host substanza list
`in`	*hostName*	Hostname to search for
`in`	*ignoreDomain*	Whether to ignore the domain portion of the hostname

This function attempts to find a substanza for the hostname contained in the "hostName" parameter in the linked list of hostnames whose head is pointed to by the "hostStanzaList" parameter.

The name is searched for literally, i.e., if it contains a domain name, then the client host entry must also contain a domain name or a wildcard pattern in order to match, except that if the [ignoreDomain] flag is TRUE, then the domain name component of the "hostName", if included as part of the name, is ignored in both the "hostName" and in all of the hostnames in the stanza.

The client host list contains one or space-separated names for the KeyString member of the hpss_cfg_stanza_t structure. Each of these is treated as a pattern, which can contain zero or more wildcard characters. For example, the entry: mda-? mda-?.sdsc.edu = { ... } would match any of the following hostnames: mda-1 mda-7 mda-3.sdsc.-edu

If the "ignoreDomain" parameter is TRUE, then the domain name part of "hostName" (if included as part of the name), is ignored in all cases.

Returns

Function returns NULL if the hostName cannot be found in any of the client host list entries, or a pointer to the substanza for that client entry if found.

Note

The first entry in the linked list which matches the pattern is returned.

9.25.3.6 int hpsscfgx_NetoptFindEntry (hpss_sockaddr_t ∗ *NetAddr*, netopt_entry_t ∗∗ *RetEntryPtr*)

Searches through the network option table, if present to find network options configured for the specified address.

Parameters

in	*NetAddr*	Network address to look up
out	*RetEntryPtr*	Returned pointer to table entry

Rumbles through the network option table, attempting to initialize the table if not already done, to find an entry that matches the specified address - returning a pointer to the table entry.

If a Default host entry was defined in the Network Options section of the HPSS.conf file, then it will always be returned if no other matching entry is found. If multiple Default entries were mistakenly configured, the first such entry will be used.

Return values

0	Found a match
-1	Did not find a match

9.25.3.7 ssize_t hpsscfgx_NetoptGetWriteSize (int *SocketDescriptor*, hpss_sockaddr_t ∗ *IpAddr*)

Return the network (TCP/IP) write size to be used for the specified connection.

Parameters

in	*Socket-Descriptor*	Connection file descriptor
in	*IpAddr*	IP Address of the connection, or 0

Attempts to find a network option entry for the specified connection (based on the local address) - if found and the entry contains a non-zero value for the network write size, that value is returned. Otherwise, the default value (based on the presence of the "HPSS_TCP_WRITESIZE" environment variable is returned).

If the IpAddr is 0 on entry, then it is determined by getting the peer address for the socket.

Returns

the write size to be used

9.25.3.8 int hpsscfgx_NetoptSetSock (int *Fd*, hpss_sockaddr_t ∗ *NetAddress*)

Adjust the Socket characteristics.

Parameters

in	*Fd*	File descriptor
in	*NetAddress*	Network address

First determine the remote network address from first NetAddress or second from the connected socket. Lookup up the network options. Use setsockopt to set the network options.

Return values

0	Socket option set
-1	Socket option could not be set

9.25.3.9 int hpsscfgx_PatternMatch (char ∗ *theString,* char ∗ *thePattern,* int ∗ *patternError*)

Pattern match function.

Parameters

in	*theString*	String to test for a pattern match
in	*thePattern*	Pattern to compare against
out	*patternError*	Set non-zero on exit if pattern error

This function implements CSH-style pattern matching for a candidate string (theString) and a pattern (thePattern). See the prologue to "matchPattern" for a description of the pattern matching.

If an error is encountered on the pattern string, ∗patternError is set non-zero.

Return values

TRUE	Pattern and string match
FALSE	Pattern and string do not match

9.26 HPSS Config File Template API

Template-based parser for config files.

Modules

- Main
- Parsers
- Validators

9.26.1 Detailed Description

Template-based parser for config files. The hpsscfgx_ReadConf functions are intended to provide a generic parser for files such as HPSS.conf, HPSS.COS, etc, which conform to the format defined for the HPSS.conf file, as described in the HPSS.conf.7 man page. These functions allow the settings to be described as a set of templates that mimic the expected hierarchical structure of the conf file itself.

The API provides functions for:

- parsing a file into a config structure using a template structure
- freeing the allocated config structure contents in accordance with the template structure

The template is as follows: name - the name of the stanza/substanza to look for at this level default_value - A value to be used if the value is not found valid - A function which tests the validity of the given value conf_ptr - The offset into the configuration structure aux - Auxilliary data used by some of the entry type functions type - A function which parses the entry into the configuration structure.

Validity functions defined in this header: HPSSCFGX_MANDATORY - The value is mandatory (The default value is accepted if one is provided and the entry doesn't exist) HPSSCFGX_OPTIONAL - The value need not exist. Not useful if a default value is given. See HPSSCFGX_MANDATORY

Parser type functions defined in this header:

HPSSCFGX_STRING - The value is a string. Copy verbatim Type: char ∗ HPSSCFGX_PORT - The value is a TCP port number in the range 0-65535 Type: in_port_t HPSSCFGX_UINT32 - The value is a 32-bit unsigned integer Type: unsigned int HPSSCFGX_FLAG - The value is a boolean flag (True or False) Type: bool HPSSCFGX_COMPOUND - The value is a substanza. Uses aux as template of substanza HPSSCFGX_IGNORE - The value should be ignored. Mainly used by HPSSCFGX_OPTIONAL validator HPSSCFGX_INVALID - The value is invalid. Mainly used by validators

API functions: hpsscfgx_ReadConf - Read a conf file hpsscfgx_ReadConfTemplate - Read and parse a conf file into a config struct hpsscfgx_DisposeConf - Free contents of a config struct hpsscfgx_FormatMessage - Format a message to indicate a parse error. Mainly useful for derived file parsers. hpsscfgx_ParseTemplate - Parse a single template entry. Mainly useful for derived file parsers.

9.27 Main

Functions

- void hpsscfgx_DisposeConf (hpsscfgx_ConfTemplate_t const ∗conf_template, void ∗conf_data)

  Dispose of a conf_data structure.
- void hpsscfgx_FormatMessage (hpss_cfg_stanza_t ∗entry, char ∗result, size_t result_sz, char const ∗msg,...)

  Convenience function to report a message about a stanza.
- int hpsscfgx_ParseTemplate (void ∗out, hpsscfgx_ConfTemplate_t const ∗tmpl, hpss_cfg_stanza_t ∗root, bool disposing, char ∗result, size_t result_sz)

  Function to convert the configuration structure to native data.
- hpss_cfg_stanza_t ∗ hpsscfgx_ReadConf (char const ∗FileName, char ∗result, size_t result_sz)

  Read a configuration file in.
- int hpsscfgx_ReadConfTemplate (char const ∗file_name, hpsscfgx_ConfTemplate_t const ∗conf_template, void ∗conf_data, char ∗result, size_t result_sz)

  Read a configuration file into a conf_data structure.

9.27.1 Detailed Description

Main template parse functions

9.27.2 Function Documentation

9.27.2.1 void hpsscfgx_DisposeConf (hpsscfgx_ConfTemplate_t const ∗ *conf_template,* void ∗ *conf_data*)

Dispose of a conf_data structure.

When the configuration data is no longer needed, this function will free its data according to the template that created it.

Parameters

in	*conf_template*	The description of the configuration data to be freed
in	*conf_data*	The configuration data to free

9.27.2.2 void hpsscfgx_FormatMessage (hpss_cfg_stanza_t ∗ *entry,* char ∗ *result,* size_t *result_sz,* char const ∗ *msg,* ...)

Convenience function to report a message about a stanza.

Parameters

in	*entry*	The entry that resulted in an error
out	*result*	The buffer to fill with the message
in	*result_sz*	The size of the output buffer
in	*msg*	The format message and following field data values

9.27.2.3 int hpsscfgx_ParseTemplate (void ∗ *out,* hpsscfgx_ConfTemplate_t const ∗ *tmpl,* hpss_cfg_stanza_t ∗ *root,* bool *disposing,* char ∗ *result,* size_t *result_sz*)

Function to convert the configuration structure to native data.

Parameters

`out`	*out*	The output configuration structure
`in`	*tmpl*	The description of the configuration
`in`	*root*	The root of the parsed config file
`in`	*disposing*	True if freeing the configuration structure
`out`	*result*	The buffer to fill with an error message
`in`	*result_sz*	The size of the output buffer

Note

Does not affect the result string if no errors.

9.27.2.4 hpss_cfg_stanza_t∗ hpsscfgx_ReadConf (char const ∗ *FileName,* char ∗ *result,* size_t *result_sz*)

Read a configuration file in.

This is a convenience function which reads the configuration file into an in-memory representation. It does not perform any validation.

Note

Does not affect result unless there is an error.

9.27.2.5 int hpsscfgx_ReadConfTemplate (char const ∗ *file_name,* hpsscfgx_ConfTemplate_t const ∗ *conf_template,* void ∗ *conf_data,* char ∗ *result,* size_t *result_sz*)

Read a configuration file into a conf_data structure.

This function parses a configuration file into a local structure, as described by the conf_template.

Parameters

`in`	*file_name*	The name of the conf file to read in
`in`	*conf_template*	The description of the structure of the stanza and configuration data
`out`	*conf_data*	The structure which will hold the parsed stanza
`out`	*result*	A text message describing any errors encountered
`in`	*result_sz*	The size of the message buffer

9.28 Parsers

Modules

- Special

    Special purpose parser functions. They are typically returned by the validator functions to indicate special behavior is required.

Functions

- int HPSSCFGX_COMPOUND (char const ∗input, void ∗out, void const ∗aux, hpss_cfg_stanza_t ∗root, bool disposing, char ∗result, size_t result_sz)

    Compound parser. Output is a pointer to a structure representing the values contained in the compound stanza. `aux` *describes the compound sub-structure. It is an array of config templates, terminated by an entry with a NULL stanza name.*
- int HPSSCFGX_ENUM (char const ∗input, void ∗out, void const ∗aux, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

    Enumeration parser. All possible choices are offered in the config template using an array of hpsscfgx_Choice-Template_t structs as aux data.
- int HPSSCFGX_ENUM_FN (char const ∗input, void ∗out, void const ∗aux, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

    Enumeration parser. All possible choices are parsed by a function which converts a string to a value. The function returns a success code.
- int HPSSCFGX_FLAG (char const ∗input, void ∗out, void const ∗aux, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

    Boolean parser. Output is a 'bool' pointer type.
- int HPSSCFGX_PORT (char const ∗input, void ∗out, void const ∗aux, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

    Port parser. Output is a 'in_port_t' pointer type.
- int HPSSCFGX_STRING (char const ∗input, void ∗out, void const ∗aux, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

    String parser. Just copies the string verbatim, allocating memory for the string.
- int HPSSCFGX_UINT32 (char const ∗input, void ∗out, void const ∗aux, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

    32-bit unsigned integer parser. Output is a 'uint32_t' pointer type
- int HPSSCFGX_UINT64 (char const ∗input, void ∗out, void const ∗aux, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

    64-bit unsigned integer parser. Output is a 'uint64_t' pointer type Values can be specified with postfixes, ie 4MB

9.28.1 Detailed Description

Parsers are called out in the config template to transform the input into a native form.

Parsers are defined with a common arg template.

input The value to parse out The given buffer to store the parsed data into. Note that this function allocates memory for the output string. Therefore, out must be a pointer to a char∗, and the char∗ must point to a valid allocation of memory or NULL. aux Auxiliary data appropriate to the particular parser entry The HPSS config stanza that triggered this filter disposing True if disposing of config data structure result The error message if parsing fails result_sz The total size of the result buffer

9.28.2 Function Documentation

9.28.2.1 int HPSSCFGX_ENUM (char const ∗ *input,* void ∗ *out,* void const ∗ *aux,* hpss_cfg_stanza_t ∗ *entry,* bool *disposing,* char ∗ *result,* size_t *result_sz*)

Enumeration parser. All possible choices are offered in the config template using an array of hpsscfgx_Choice-Template_t structs as aux data.

Note

Note that comparisons with enum names are case insensitive.

9.28.2.2 int HPSSCFGX_STRING (char const ∗ *input,* void ∗ *out,* void const ∗ *aux,* hpss_cfg_stanza_t ∗ *entry,* bool *disposing,* char ∗ *result,* size_t *result_sz*)

String parser. Just copies the string verbatim, allocating memory for the string.

If the AUX pointer is not NULL, it is expected to be a hpsscfgx_StringLimit_t type which sets the string's limits.

9.29 Special

Special purpose parser functions. They are typically returned by the validator functions to indicate special behavior is required.

Functions

- int HPSSCFGX_IGNORE (char const ∗input, void ∗out, void const ∗aux, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

  Ignores this configuration parameter.
- int HPSSCFGX_INVALID (char const ∗input, void ∗out, void const ∗aux, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

  Declare this configuration parameter invalid.

9.29.1 Detailed Description

Special purpose parser functions. They are typically returned by the validator functions to indicate special behavior is required.

9.30 Validators

Functions

- custom_func_t HPSSCFGX_MANDATORY (char const ∗section, void ∗conf, custom_func_t type, char const ∗default_value, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

  Marks the section as mandatory.
- custom_func_t HPSSCFGX_OPTIONAL (char const ∗section, void ∗conf, custom_func_t type, char const ∗default_value, hpss_cfg_stanza_t ∗entry, bool disposing, char ∗result, size_t result_sz)

  Marks the section as optional.

9.30.1 Detailed Description

Validators are called out in the config template to make sure the configuration is valid. They can declare that certain items are mandatory, that items should be ignored, etc. The validator will return a parser which enforces the chosen policy.

Validators are defined with a common arg template.

section The name of the section being parsed conf Pointer to the template configuration for this point. type The proposed parse type for this stanza entry The current stanza under consideration disposing True if disposing of config data structure result The error message if validation fails result_sz The total size of the result buffer

9.31 POSIX APIs

Posix Compliant HPSS APIs (Unsupported)

Functions

- int hpss_CreateWithHints (char ∗Path, mode_t Mode, hpss_cos_hints_t ∗HintsIn, hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

    Create an HPSS file with hints.
- int hpss_OpenWithHints (char ∗Path, int Oflag, mode_t Mode, hpss_cos_hints_t ∗HintsIn, hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

    Open a file using HPSS hints.
- int hpss_PosixCreate (char ∗Path, mode_t Mode)

    Create an HPSS file.
- int hpss_PosixFstat (int Fildes, struct stat ∗Buf)

    Retrieve information about an open file or directory.
- long hpss_PosixLseek (int Fildes, long Offset, int Whence)

    Set the file offset for an open file handl.
- int hpss_PosixLstat (char ∗Path, struct stat ∗Buf)

    Obtain information about a file or directory, without traversing symbolic links.
- int hpss_PosixOpen (char ∗Path, int Oflag, mode_t Mode)

    Open an HPSS file.
- int hpss_PosixStat (char ∗Path, struct stat ∗Buf)

    Obtain information about a file or directory.

9.31.1 Detailed Description

Posix Compliant HPSS APIs (Unsupported)

9.31.2 Function Documentation

9.31.2.1 int hpss_CreateWithHints (char ∗ *Path,* mode_t *Mode,* hpss_cos_hints_t ∗ *HintsIn,* hpss_cos_priorities_t ∗ *HintsPri,* hpss_cos_hints_t ∗ *HintsOut*)

Create an HPSS file with hints.

Parameters

`in`	*Path*	Path to file to be opened
`in`	*Mode*	Desired file permissions, if created
`in`	*HintsIn*	Desired class of service
`in`	*HintsPri*	Priorities of hints
`out`	*HintsOut*	Granted class of service

The 'hpss_CreateWithHints' function creates a file specified by by *Path*, with permissions as specified by *Mode* and using the class of service values specified by *HintsIn* and *HintsPri*, if non-NULL.

Return values

0	Success
Other	Negated POSIX error code

9.31.2.2 int hpss_OpenWithHints (char * *Path,* int *Oflag,* mode_t *Mode,* hpss_cos_hints_t * *HintsIn,* hpss_cos_priorities_t * *HintsPri,* hpss_cos_hints_t * *HintsOut*)

Open a file using HPSS hints.

Parameters

`in`	*Path*	Path to file to be opened
`in`	*Oflag*	Type of file access • O_RDONLY • O_WRONLY • O_RDWR • O_APPEND • O_TRUNC
`in`	*Mode*	Desired file perms if created; if O_CREATE is specified and a file is created, this argument gives the file mode used for determining the file mode for the created file
`in`	*HintsIn*	Desired class of service
`in`	*HintsPri*	Desired priority
`out`	*HintsOut*	Granted class of service

The 'hpss_OpenWithHints' function establishes a connection between a file, named by 'Path', and a file handle.

Return values

>=0	Opened file handle
<0	Negated POSIX error code

9.31.2.3 int hpss_PosixCreate (char * *Path,* mode_t *Mode*)

Create an HPSS file.

Parameters

`in`	*Path*	Path to file to be opened
`in`	*Mode*	Desired file perms if create

The 'hpss_PosixCreate' function creates a file specified by by *Path*, with permissions as specified by *Mode*

Return values

>=0	Open file handle
<0	Negated POSIX error code

9.31.2.4 int hpss_PosixFstat (int *Fildes,* struct stat * *Buf*)

Retrieve information about an open file or directory.

Parameters

`in`	*Fildes*	Open file or directory to retrieve information for
`out`	*Buf*	Returned statistics

Obtains information about the open file or directory *Fildes* and returns the information in a structure pointed to by *Buf*.

Return values

0	Success
Other	Negated POSIX error code

9.31.2.5 long hpss_PosixLseek (int *Fildes,* long *Offset,* int *Whence*)

Set the file offset for an open file handl.

Parameters

`in`	*Fildes*	File descriptor of open object
`in`	*Offset*	Number of bytes to calculate new offset
`in`	*Whence*	Origin for the seek

This function sets the file offset for the open file handle, ’Fildes’.

Parameters

`in`	*Fildes*	Open file handle for which the file offset is set
`in`	*Offset*	Number of bytes to be used in calculating the new file offset; dependent upon the value of *Whence* as to the final effect on the new file offset.
`in`	*Whence*	SEEK_SET, SEEK_CUR, or SEEK_END

Whence can be one of three values: SEEK_SET, where the file offset is set to *Offset*, SEEK_CUR, where the file offset is incremented by *Offset*, or SEEK_END, where the file offset is set to the current end of file plus *Offset*.

Return values

>=0	The resulting offset measured in bytes from the beginning of the file
<0	Negated POSIX error code

9.31.2.6 int hpss_PosixLstat (char ∗ *Path,* struct stat ∗ *Buf*)

Obtain information about a file or directory, without traversing symbolic links.

Parameters

`in`	*Path*	Path of file to retrieve information for
`out`	*Buf*	Returned statistics

The ’hpss_PosixLStat’ function obtains information about the file or directory named by *Path* and returns the information in a structure pointed to by *Buf*. If *Path* refers to a symbolic link, the information about the link itself will be returned.

Return values

0	Success
Other	Negated POSIX error code

9.31.2.7 int hpss_PosixOpen (char ∗ *Path,* int *Oflag,* mode_t *Mode*)

Open an HPSS file.

Parameters

`in`	*Path*	Path to file to be opened
`in`	*Oflag*	Type of file access
`in`	*Mode*	Desired file perms if create

The 'hpss_PosixOpen' function establishes a connection between a file, named by *Path*, and a file handle.

Oflag File status and file access modes to be assigned: Applicable values are: O_RDONLY, O_WRONLY, O_RDW-R, O_APPEND, O_CREAT, O_EXCL, O_TRUNC *Mode* If O_CREAT is specified and a file is created, this argument gives the file mode used for determining the file mode for the created file.

Return values

>=0	Opened file handle
<0	POSIX error code

Note

The HPSS_COS environment variable can be set to specify the HPSS class of service.

9.31.2.8 int hpss_PosixStat (char * *Path,* struct stat * *Buf*)

Obtain information about a file or directory.

Parameters

`in`	*Path*	Path of file to retrieve information for
`out`	*Buf*	Returned statistics

The 'hpss_PosixStat' function obtains information about the file or directory named by *Path* and returns the information in a structure pointed to by *Buf*. If *Path* refers to a symbolic link, the link will be traversed to get to the target object.

Return values

0	Success
Other	Negated POSIX error code

9.32 Data Containers

Functions which implement data containers.

Files

- file hpss_ihashtable.h

  Generic intrusive hash table.
- file hpss_ilist.h

  Generic intrusive doubly-linked circular list.
- file hpss_irbtree.c

  Generic intrusive red-black tree.
- file hpss_irbtree.h

  Generic intrusive red-black tree.

Classes

- struct hpss_irbtree

  A root structure for the tree. More...
- struct hpss_irbtree_node

  A node in the tree. More...

Macros

- #define hpss_irbtree_container(ptr, type, member) ((type∗)(((char∗)ptr) - offsetof(type, member)))

  Get the container of a node.
- #define LEFT 0

  Index for left child.
- #define RIGHT 1

  Index for right child.

Typedefs

- typedef hpss_ilist_t ∗(∗ hpss_ilist_alloc_func_t)(hpss_reqid_t RequestID, size_t NumBytes, const char ∗File-Name, const char ∗FunctionName, const int LineNumber)

  Memory allocator for custom list initialization.
- typedef void(∗ hpss_ilist_destructor_t)(hpss_ilist_node_t ∗Node)

  Node destructor.
- typedef struct
  hpss_ilist_iterator_token ∗ hpss_ilist_iterator_token_h

  Typedef for opaque token.
- typedef
  hpss_ilist_foreach_retval_e(∗ hpss_ilist_mapped_func_t)(hpss_ilist_node_t ∗Node)

  Mapping function type.
- typedef struct hpss_ilist_node hpss_ilist_node_t

  Just the typedef for the following hpss_ilist_node struct.
- typedef struct hpss_ilist hpss_ilist_t

  List type.
- typedef enum hpss_irbtree_color hpss_irbtree_color_t

Node colors.

- typedef int(∗ hpss_irbtree_node_callback_t)(const hpss_irbtree_node_t ∗Node, void ∗Arg)

  Node callback.
- typedef int(∗ hpss_irbtree_node_comparator_t)(const hpss_irbtree_node_t ∗lhs, const hpss_irbtree_node_t ∗rhs)

  Node comparator.
- typedef void(∗ hpss_irbtree_node_destructor_t)(hpss_irbtree_node_t ∗Node)

  Node destructor.
- typedef void(∗ hpss_irbtree_node_printer_t)(const hpss_irbtree_node_t ∗Node)

  Callback to print a node.
- typedef struct hpss_irbtree_node hpss_irbtree_node_t

  Typedef for struct hpss_irbtree_node.
- typedef struct hpss_irbtree hpss_irbtree_t

  Typedef for struct hpss_irbtree.

Enumerations

- enum hpss_ihashtable_constants { DEFAULT_HASHTABLE_SIZE = 100, MAX_LOAD_FACTOR = 150 }

  Constant values for the intrusive hash table container.
- enum hpss_irbtree_color { IRBTREE_RED = 0, IRBTREE_BLACK = 1 }

  Node colors.

Functions

- int32_t hpss_ilist_append (hpss_ilist_t ∗List, hpss_ilist_node_t ∗Node)

  Add a node to the end of an existing list.
- void hpss_ilist_clear (hpss_ilist_t ∗List)

  Clear an entire list by taking out all nodes and freeing memory.
- hpss_ilist_t ∗ hpss_ilist_custom_init (hpss_ilist_destructor_t Destroy, hpss_ilist_alloc_func_t AllocFunc, hpss_reqid_t RequestID, const char ∗FileName, const char ∗FunctionName, const int LineNumber)

  Initialize a list with a custom memory allocator.
- bool hpss_ilist_empty (hpss_ilist_t ∗List)

  Check whether a list node points to anything else.
- void hpss_ilist_foreach (hpss_ilist_t ∗List, hpss_ilist_mapped_func_t F)

  Map a function across all nodes in a list.
- void hpss_ilist_foreach_range (hpss_ilist_t ∗List, hpss_ilist_mapped_func_t F, int32_t StartIndex, int32_t EndIndex)

  Map a function across all nodes in a given range of a list.
- hpss_ilist_node_t ∗ hpss_ilist_get_current (const hpss_ilist_t ∗List)

  Return the current node in the list, useful for starting iteration.
- hpss_ilist_node_t ∗ hpss_ilist_get_current_r (const hpss_ilist_iterator_token_h Token)

  Return the current node in the list, useful for starting reentrant iteration.
- hpss_ilist_node_t ∗ hpss_ilist_get_node (hpss_ilist_t ∗List, int32_t Index)

  Get a node at a given index in a list.
- int32_t hpss_ilist_get_size (const hpss_ilist_t ∗const List)

  Access the size variable in the private hpss_ilist_t struct.
- void hpss_ilist_halt_iterator (hpss_ilist_t ∗List)

  Halt iteration, reset Start, Current, and End to NULL.
- hpss_ilist_t ∗ hpss_ilist_init (hpss_ilist_destructor_t Destroy)

  Initialize a list.

- int32_t hpss_ilist_init_iterator (hpss_ilist_t ∗List)

  Init the iterator to iterate over the entire list.
- hpss_ilist_iterator_token_h hpss_ilist_init_iterator_r (hpss_ilist_t ∗List)

  Define a range of indices to include during iteration. Also initialize token for reentrant iteration.
- int32_t hpss_ilist_init_iterator_range (hpss_ilist_t ∗List, int32_t Start, int32_t End)

  Init the Current, Start, and End variables in the hpss_ilist struct by giving a range of indices to iterate over.
- hpss_ilist_iterator_token_h hpss_ilist_init_iterator_range_r (hpss_ilist_t ∗List, int32_t Start, int32_t End)

  Tell the iterator what range of indices to include. Also initialize token for reentrant iteration.
- int32_t hpss_ilist_insert (hpss_ilist_t ∗List, hpss_ilist_node_t ∗Node1, hpss_ilist_node_t ∗Node2)

  Insert a node into an existing list.
- int32_t hpss_ilist_join (hpss_ilist_t ∗List1, hpss_ilist_t ∗List2)

  Attach one list to the end of another list.
- int32_t hpss_ilist_prepend (hpss_ilist_t ∗List, hpss_ilist_node_t ∗Node)

  Add a node to the start of an existing list.
- int32_t hpss_ilist_reinit_iterator (hpss_ilist_t ∗List)

  Reinitialize iteration on the entire list using the halt and initialize functions above.
- int32_t hpss_ilist_reinit_iterator_range (hpss_ilist_t ∗List, int32_t Start, int32_t End)

  Reinitialize iteration on a range of indices using the halt and initialize functions above.
- void hpss_ilist_remove (hpss_ilist_t ∗List, hpss_ilist_node_t ∗Node)

  Remove a list node from a list.
- void hpss_irbtree_clear (hpss_irbtree_t ∗Tree, hpss_irbtree_node_destructor_t Destructor)

  Destroy an entire tree.
- int hpss_irbtree_empty (const hpss_irbtree_t ∗Tree)

  Check if a tree is empty.
- hpss_irbtree_node_t ∗ hpss_irbtree_find (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node)

  Find a node in a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_find_cmp (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node, hpss_irbtree_node_comparator_t Comparator, int KeyCompare)

  Find a node in a tree.
- int hpss_irbtree_foreach (const hpss_irbtree_t ∗Tree, hpss_irbtree_node_callback_t Callback, void ∗CallbackArg)

  Call a function on each node in a tree.
- int hpss_irbtree_foreach_node (const hpss_irbtree_node_t ∗Begin, const hpss_irbtree_node_t ∗End, hpss_-irbtree_node_callback_t Callback, void ∗CallbackArg)

  Call a function on a range of nodes by forward iteration.
- int hpss_irbtree_foreach_node_prev (const hpss_irbtree_node_t ∗RBegin, const hpss_irbtree_node_t ∗R-End, hpss_irbtree_node_callback_t Callback, void ∗CallbackArg)

  Call a function on a range of nodes by reverse iteration.
- void hpss_irbtree_init (hpss_irbtree_t ∗Tree, hpss_irbtree_node_comparator_t Comparator)

  Initialize a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_insert (hpss_irbtree_t ∗Tree, hpss_irbtree_node_t ∗Node)

  Insert a node into a tree.
- void hpss_irbtree_insert_multi (hpss_irbtree_t ∗Tree, hpss_irbtree_node_t ∗Node)

  Insert a (possibly) duplicate node into a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_lower_bound (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node)

  Find the lower bound for a key.
- hpss_irbtree_node_t ∗ hpss_irbtree_max (const hpss_irbtree_t ∗Tree)

  Get the right-most object in the tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_min (const hpss_irbtree_t ∗Tree)

  Get the left-most object in the tree.

- hpss_irbtree_node_t ∗ hpss_irbtree_node_next (const hpss_irbtree_node_t ∗Node)

  Iterate to the next node in the tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_node_prev (const hpss_irbtree_node_t ∗Node)

  Iterate to the previous node in the tree.
- void hpss_irbtree_print (const hpss_irbtree_t ∗Tree, hpss_irbtree_node_printer_t Printer)

  Call a printer function on each node in a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_remove (hpss_irbtree_t ∗Tree, hpss_irbtree_node_t ∗Node, hpss_-irbtree_node_destructor_t Destructor)

  Remove a node from a tree.
- size_t hpss_irbtree_size (const hpss_irbtree_t ∗Tree)

  Get number of nodes in the tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_upper_bound (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node)

  Find the upper bound for a key.

9.32.1 Detailed Description

Functions which implement data containers.

9.32.2 Class Documentation

9.32.2.1 struct hpss_irbtree

A root structure for the tree.

Class Members

hpss_irbtree_-node_-comparator_t	Comparator	Comparator for tree
hpss_irbtree_-node_t ∗	Root	Pointer to tree's root node
size_t	Size	Number of nodes in the tree

9.32.2.2 struct hpss_irbtree_node

A node in the tree.

Note

The parent pointer and node color are mashed up in a single variable. This is taking advantage of the assumption that tree nodes are at least halfword-aligned, meaning the bottom bit is always 0 in its address. This leaves room to store the color (which only needs one bit) in the bottom bit. This helps to save at least one byte per node, but in practice it will probably save a long word per node due to other alignment constraints. This means a potential savings of 25% for node memory overhead.
The Child pointers are an array to take advantage of implementation symmetry. This helps reduce a lot of code because much of it is simply mirror copies (exchange all LEFT for RIGHT and vice versa).

Class Members

hpss_irbtree_-node_t *	Child[2]	Pointers to node's children
uintptr_t	ParentColor	Mashup of parent pointer and node color

9.32.3 Macro Definition Documentation

9.32.3.1 #define hpss_irbtree_container(*ptr, type, member*) ((type∗)(((char∗)ptr) - offsetof(type, member)))

Get the container of a node.

This is essentially a *container_of* macro.

Parameters

`in`	*ptr*	Pointer to bfs_TreeNode_t
`in`	*type*	Type of the struct containing the bfs_TreeNode_t
`in`	*member*	Member of *type* that *ptr* points to

Note

You should only use this when you absolutely know what struct this tree node belongs to.

9.32.4 Typedef Documentation

9.32.4.1 typedef **hpss_ilist_t**∗(∗ hpss_ilist_alloc_func_t)(**hpss_reqid_t** RequestID, **size_t** NumBytes, const char ∗FileName, const char ∗FunctionName, const int LineNumber)

Memory allocator for custom list initialization.

Functions of this type are passed to hpss_ilist_custom_init() so calling code can specify a method of dynamically allocating the memory for the hpss_ilist_t

See Also

hpss_ilist_custom_init()

9.32.4.2 typedef void(∗ hpss_ilist_destructor_t)(**hpss_ilist_node_t** ∗Node)

Node destructor.

Use this as a callback to cleanup the struct containing a node

See Also

hpss_ilist_clear

9.32.4.3 typedef struct hpss_ilist_iterator_token∗ **hpss_ilist_iterator_token_h**

Typedef for opaque token.

Pass this token to methods that handle reentrant iteration

Note

The struct definition is private as there is no need for calling code to manipulate the internals of the token

9.32.4.4 typedef **hpss_ilist_foreach_retval_e**(∗ hpss_ilist_mapped_func_t)(**hpss_ilist_node_t** ∗Node)

Mapping function type.

This is a function pointer that should be used whenever you want to declare a function that should be mapped across nodes by the hpss_ilist_foreach() function

Note

To declare a function of this type, you need not use the hpss_ilist_mapped_func_t keyword, just make a function that takes an hpss_ilist_node_t∗ node and returns KEEP_ITERATING or STOP_ITERATING

See Also

hpss_ilist_foreach

9.32.4.5 typedef struct hpss_ilist **hpss_ilist_t**

List type.

List object that acts as a handle for a list to be passed to hpss_ilist methods and encapsulates useful info about the list such as the size of list, iterator state (Start, Current, and End nodes), and destructor for nodes

9.32.4.6 typedef int(∗ hpss_irbtree_node_callback_t)(const **hpss_irbtree_node_t** ∗Node, void ∗Arg)

Node callback.

Use this as a callback on a node. It should return non-zero for early termination.

See Also

hpss_irbtree_foreach
hpss_irbtree_foreach_node
hpss_irbtree_foreach_node_prev

9.32.4.7 typedef int(∗ hpss_irbtree_node_comparator_t)(const **hpss_irbtree_node_t** ∗lhs, const **hpss_irbtree_node_t** ∗rhs)

Node comparator.

Use this as a callback to compare two nodes. It should return $<$0 for lhs $<$ rhs, 0 for lhs == rhs, and $>$0 for lhs $>$ rhs.

You should never change objects in a tree in such a way that breaks the strict ordering. If this is necessary, you must first remove the object from the tree, update it, and re-insert it.

See Also

hpss_irbtree_init
hpss_irbtree_find_cmp

9.32.4.8 typedef void(∗ hpss_irbtree_node_destructor_t)(**hpss_irbtree_node_t** ∗Node)

Node destructor.

Use this as a callback to cleanup the struct containing a node

See Also

hpss_irbtree_remove
hpss_irbtree_clear

9.32.4.9 typedef void(∗ hpss_irbtree_node_printer_t)(const **hpss_irbtree_node_t** ∗Node)

Callback to print a node.

See Also

hpss_irbtree_print

9.32.5 Enumeration Type Documentation

9.32.5.1 enum **hpss_ihashtable_constants**

Constant values for the intrusive hash table container.

Enumerator

DEFAULT_HASHTABLE_SIZE	The default hash table size.
MAX_LOAD_FACTOR	A table will be enlarged once it hits this threshold.

9.32.5.2 enum **hpss_irbtree_color**

Node colors.

Enumerator

IRBTREE_RED	The node is colored red
IRBTREE_BLACK	The node is colored black

9.32.6 Function Documentation

9.32.6.1 int32_t hpss_ilist_append (**hpss_ilist_t** ∗ *List,* **hpss_ilist_node_t** ∗ *Node*)

Add a node to the end of an existing list.

This is useful for implementing queues

Parameters

`in`	*List*	List to append to
`in`	*Node*	Node to append

Returns

-1 if the insertion operation did not succeed, 0 otherwise

9.32.6.2 void hpss_ilist_clear (**hpss_ilist_t** ∗ *List*)

Clear an entire list by taking out all nodes and freeing memory.

This removes each node from a list and calls *Destructor* on it.

Note

List is reinitialized at the end of this method, ready to be used again

Parameters

`in`	*List*	List to destroy
`in`	*Destructor*	Callback to use on each node

9.32.6.3 **hpss_ilist_t**∗ hpss_ilist_custom_init (**hpss_ilist_destructor_t** *Destroy,* **hpss_ilist_alloc_func_t** *AllocFunc,* **hpss_reqid_t** *RequestID,* const char ∗ *FileName,* const char ∗ *FunctionName,* const int *LineNumber*)

Initialize a list with a custom memory allocator.

This uses a memory allocator supplied by the calling code to create an hpss_ilist_t. Similar to hpss_ilist_init(), this function makes the sentinel node point to itself and assigns member variables such as Size and Destructor.

Returns

List that we just initialized

Parameters

`in`	*Destroy*	Destructor function that shows how to get rid of nodes. Destroy can be NULL if desired. This would be useful if you wish to use hpss_ilist_remove in order to take a node out of a list but not destroy it so it can still be used later.
`in`	*AllocFunc*	Allocator function that defines how to dynamically allocate the memory for the hpss_ilist_t struct. If the calling code does not require a specific allocator, use the generic hpss_ilist_init() function instead.
`in`	*RequestID*	A request ID of type hpss_reqid_t for logging purposes
`in`	*FunctionName*	The name of the function that called this init function, also for logging purposes
`in`	*FileName*	The name of the file from which hpss_ilist_custom_init() was called, also for logging purposes
`in`	*LineNumber*	The line number where hpss_ilist_custom_init() was called, also for logging purpose

9.32.6.4 bool hpss_ilist_empty (**hpss_ilist_t** ∗ *List*)

Check whether a list node points to anything else.

If the list node points to itself, it is "empty", i.e. there is no linkage to other list nodes. A list with just a sentinel node is empty and hpss_ilist_empty will return True.

Parameters

in	*List*	List to check

Returns

True if List is empty, false if List is not empty

9.32.6.5 void hpss_ilist_foreach (hpss_ilist_t ∗ *List,* hpss_ilist_mapped_func_t *F*)

Map a function across all nodes in a list.

Given a pointer to the top of the list and a function, call the function on each node. This is a linear time function.

Parameters

in	*List*	The top of the list, which has an empty payload
in	*F*	A function of type hpss_ilist_mapped_func_t, takes a pointer to an hpss_ilist_-node_t as its only argument

Note

First node is skipped as always because its payload is empty
User must define an operation to be mapped in the form of a function, F. Default mapping functions may be provided

9.32.6.6 void hpss_ilist_foreach_range (hpss_ilist_t ∗ *List,* hpss_ilist_mapped_func_t *F,* int32_t *StartIndex,* int32_t *EndIndex*)

Map a function across all nodes in a given range of a list.

Given a pointer to the top of the list, a function, and a range of indices, call the function on each node. This is a linear time function.

Parameters

in	*List*	A pointer to the hpss_ilist_t we will be mapping a function over
in	*F*	A function of type hpss_ilist_mapped_func_t, takes a pointer to an hpss_ilist_-node_t as its only argument
in	*StartIndex*	Which node to start with
in	*EndIndex*	Which node to end with

Note

Ranges are inclusive, so nodes at StartIndex and EndIndex will be passed to function F
User must define an operation to be mapped in the form of a function, F. Default mapping functions may be provided
If EndIndex is larger than the length of the list, we will stop when we reach the sentinel node (end of list)
hpss_ilist_foreach_range(List, func, 0, size_of_list-1) is the same as hpss_ilist_foreach(List, func)

9.32.6.7 hpss_ilist_node_t∗ hpss_ilist_get_current (const hpss_ilist_t ∗ *List*)

Return the current node in the list, useful for starting iteration.

Parameters

in	*List*	Pointer to a list structure

Returns

The current node stored in List->Current, NULL if the List is empty

9.32.6.8 hpss_ilist_node_t* hpss_ilist_get_current_r (const hpss_ilist_iterator_token_h *Token*)

Return the current node in the list, useful for starting reentrant iteration.

Parameters

in	*Token*	Token that has state information for reentrant iteration

Returns

The current node stored in Token->Current, NULL if the List is empty

9.32.6.9 hpss_ilist_node_t* hpss_ilist_get_node (hpss_ilist_t * *List,* int32_t *Index*)

Get a node at a given index in a list.

Iterate through the given list and return the node at given index

Parameters

in	*List*	Pointer to a list structure
in	*Index*	An integer index at which we should stop and return

Returns

The address of the node at Index in List

Note

If the node does not belong to the list, return a NULL pointer
Index 0 is the first node with data, which is Sentinel->Next
Even though the sentinel node is in the list, it does not get an index as there is no reason to access it externally and it doesn't contribute to List->Size

9.32.6.10 int32_t hpss_ilist_get_size (const hpss_ilist_t *const *List*)

Access the size variable in the private hpss_ilist_t struct.

Parameters

in	*List*	an hpss_ilist_t whose size we want to return

Returns

size of the list as a 4 byte integer

9.32.6.11 void hpss_ilist_halt_iterator (**hpss_ilist_t** ∗ *List*)

Halt iteration, reset Start, Current, and End to NULL.

Use this function when an iterator has been initialized, but the calling code no longer needs to finish iterating over the entire range defined by the calling code. In order to iterate over the list in the future, the iterator needs all its state variables (Start, Current, and End) to be NULL again. hpss_ilist_init_iterator() will return a non-zero error code if the Start, Current, and/or End variables are non-NULL. A non-NULL value means that the list is currently undergoing iteration. Unless a developer uses the reentrant iterator, a list can not support multiple sources of iteration happening simultaneously. Thus, a developer needs to call the halt method if there is no longer a need to complete the current iteration. Call the halt method BEFORE trying to call hpss_ilist_init_iterator() again. The initialization won't be successful unless the previous iteration has been completed (meaning the whole range has been iterated over) or hpss_ilist_halt_iterator() is called.

Parameters

`in`	*List*	A List pointer of type hpss_ilist_t∗

9.32.6.12 **hpss_ilist_t**∗ hpss_ilist_init (**hpss_ilist_destructor_t** *Destroy*)

Initialize a list.

This simply makes the sentinel node point to itself and assigns member variables such as Size and Destructor

Parameters

`in`	*Destroy*	a destructor of type hpss_ilist_destructor_t∗

Returns

List List that was initialized

9.32.6.13 int32_t hpss_ilist_init_iterator (**hpss_ilist_t** ∗ *List*)

Init the iterator to iterate over the entire list.

Parameters

`in`	*List*	A List pointer of type hpss_ilist_t∗

Returns

0 for success, HPSS_E_EXIST for failure

9.32.6.14 **hpss_ilist_iterator_token_h** hpss_ilist_init_iterator_r (**hpss_ilist_t** ∗ *List*)

Define a range of indices to include during iteration. Also initialize token for reentrant iteration.

Parameters

`in`	*List*	A List pointer of type hpss_ilist_t∗

Returns

The token to be passed into reentrant methods

Note

Reentrant iteration is read-only. Do not alter the contents of the list while iterating because other threads will not know of the adjustments

9.32.6.15 int32_t hpss_ilist_init_iterator_range (hpss_ilist_t ∗ *List*, int32_t *Start*, int32_t *End*)

Init the Current, Start, and End variables in the hpss_ilist struct by giving a range of indices to iterate over.

Note

This function allows specification of a range of indices. Use 0 and List->Size-1 to iterate over the entire list. This function does accept negative indices to refer to the nodes close to the end the list. However, forward iteration will not complete if Start < 0 and End >= 0. Similarly, backward iteration will not complete if Start >= 0 and End < 0. Once we reach the Sentinel node, we end iteration.

Parameters

`in`	*List*	A List pointer of type hpss_ilist_t∗
`in`	*Start*	Index of first node in the list we will iterate over
`in`	*End*	Index of last node to be included in iteration

Returns

0 for success, HPSS_E_EXIST for failure

9.32.6.16 hpss_ilist_iterator_token_h hpss_ilist_init_iterator_range_r (hpss_ilist_t ∗ *List*, int32_t *Start*, int32_t *End*)

Tell the iterator what range of indices to include. Also initialize token for reentrant iteration.

Parameters

`in`	*List*	A List pointer of type hpss_ilist_t∗
`in`	*Start*	First node to in the sublist we will iterate over
`in`	*End*	Last node to be included in iteration

Returns

The token to be passed into reentrant methods

Note

Reentrant iteration is read-only. Do not alter the contents of the list while iterating because other threads will not know of the adjustments

9.32.6.17 int32_t hpss_ilist_insert (hpss_ilist_t ∗ *List*, hpss_ilist_node_t ∗ *Node1*, hpss_ilist_node_t ∗ *Node2*)

Insert a node into an existing list.

This is useful for implementing sorted lists Insert Node2 after Node1 in List

Parameters

`in`	*List*	List to insert node into
`in`	*Node1*	Node that is right before where we want Node2 to go
`in`	*Node2*	Node to add to List

Returns

-1 if the insertion operation did not succeed, 0 otherwise

Warning

Do not insert a node into a list it is already part of; you will end up with an orphan node

9.32.6.18 int32_t hpss_ilist_join (**hpss_ilist_t** ∗ *List1,* **hpss_ilist_t** ∗ *List2*)

Attach one list to the end of another list.

Given two lists, tack the nodes of the second list onto the end of the first

Parameters

`in`	*List1*	List pointer, this will be the list that becomes longer
`in`	*List2*	List pointer, this list will become empty and get reinitialized

Returns

-1 if the lists could not be joined, 0 otherwise

9.32.6.19 int32_t hpss_ilist_prepend (**hpss_ilist_t** ∗ *List,* **hpss_ilist_node_t** ∗ *Node*)

Add a node to the start of an existing list.

This is useful for implementing stacks

Parameters

`in`	*List*	List to prepend to
`in`	*Node*	Node to prepend

Returns

-1 if the insertion operation did not succeed, 0 otherwise

9.32.6.20 int32_t hpss_ilist_reinit_iterator (**hpss_ilist_t** ∗ *List*)

Reinitialize iteration on the entire list using the halt and initialize functions above.

This function is a wrapper function that calls hpss_ilist_init_iterator(), logs any error that came from the init, then calls hpss_ilist_halt_iterator() and hpss_ilist_init_iterator() if the first attept at intializing returned a nonzero error code.

Parameters

in	*List*	A List pointer of type hpss_ilist_t*

Returns

HPSS_E_ALREADY if the List had a previous iterator set that had to be cancelled before a new iteration could begin, HPSS_E_NOERROR otherwise

9.32.6.21 int32_t hpss_ilist_reinit_iterator_range (**hpss_ilist_t** * *List,* int32_t *Start,* int32_t *End*)

Reinitialize iteration on a range of indices using the halt and initialize functions above.

This function is a wrapper function that calls hpss_ilist_init_iterator(), logs any error that came from the init, then calls hpss_ilist_halt_iterator() and hpss_ilist_init_iterator() if the first attept at intializing returned a nonzero error code.

Parameters

in	*List*	A List pointer of type hpss_ilist_t*
in	*Start*	The index of the first node to include in the iteration
in	*End*	The index of the last node to include in the iteration

Returns

HPSS_E_ALREADY if the List had a previous iterator set that had to be cancelled before a new iteration could begin, HPSS_E_NOERROR otherwise

9.32.6.22 void hpss_ilist_remove (**hpss_ilist_t** * *List,* **hpss_ilist_node_t** * *Node*)

Remove a list node from a list.

This will remove a single node from a list, joining its two ends together

Note

If you remove List->Current, it will be updated to its Next node. If the Next node is the Sentinel, that means we just got to the end of the list. In that case, iteration should halt, so we'll reinit the iterator by setting all iterator member variables to be NULL.

Parameters

in	*List*	List pointer
in	*Node*	Node to remove

9.32.6.23 void hpss_irbtree_clear (**hpss_irbtree_t** * *Tree,* **hpss_irbtree_node_destructor_t** *Destructor*)

Destroy an entire tree.

This calls *Destructor* on every node in *Tree* and resets it to an empty state.

Parameters

in	*Tree*	Tree to destroy
in	*Destructor*	Callback to use on every node in the tree

9.32.6.24 int hpss_irbtree_empty (const **hpss_irbtree_t** ∗ *Tree*)

Check if a tree is empty.

Parameters

in	*Tree*	Tree to check for emptiness

Returns

whether the tree is empty

9.32.6.25 **hpss_irbtree_node_t**∗ hpss_irbtree_find (const **hpss_irbtree_t** ∗ *Tree,* const **hpss_irbtree_node_t** ∗ *Node*)

Find a node in a tree.

Parameters

in	*Tree*	Tree to search
in	*Node*	Node to compare

Returns

matching node from tree
NULL for no match

Note

Node should point to a node inside a struct that has sufficient data for *Comparator* to perform a valid comparison. This means you can either use a dummy node or a real node, and it does not have to actually be in the tree. This will return the left-most node in the tree where *Comparator* returns 0. This allows you to repeatedly use *hpss_irbtree_node_next* to get all objects that would return 0 from *Comparator*.
For example:

```
struct myStruct x;
x.ComparisonData = Criteria;

hpss_irbtree_node_t *node = hpss_irbtree_find(&Tree, &x.TreeNode);
while(node != NULL && myComparator(&x.TreeNode, node) == 0)
{
  struct myStruct *ptr = hpss_irbtree_container(node, struct myStruct, TreeNode);
  // do something with ptr
  node = hpss_irbtree_node_next(node);
}
```

9.32.6.26 **hpss_irbtree_node_t**∗ hpss_irbtree_find_cmp (const **hpss_irbtree_t** ∗ *Tree,* const **hpss_irbtree_node_t** ∗ *Node,* **hpss_irbtree_node_comparator_t** *Comparator,* int *KeyCompare*)

Find a node in a tree.

Similar to hpss_irbtree_find, but using alternate comparator.

Parameters

in	*Tree*	Tree to search
in	*Node*	Node to compare
in	*Comparator*	Alternate comparator
in	*KeyCompare*	Whether *Comparator* is a subset of the tree's comparator

Returns

matching node from tree
NULL for no match

Note

If *KeyCompare* is non-zero, then this is still a binary search. It will only work if *Comparator* is a subset of the tree's comparator, i.e. it has the following constraints:

```
if   (*Comparator)(n1, n2) < 0
then (*Tree->Comparator)(n1, n2) < 0

if   (*Comparator)(n1, n2) > 0
then (*Tree->Comparator)(n1, n2) > 0
```

Warning

If *KeyCompare* is zero, then this is a O(n) search. You may want to consider using multiple trees with differing comparators to maintain O(log n) search time.

9.32.6.27 int hpss_irbtree_foreach (const **hpss_irbtree_t** ∗ *Tree,* **hpss_irbtree_node_callback_t** *Callback,* void ∗ *CallbackArg*)

Call a function on each node in a tree.

Parameters

in	*Tree*	Tree to print
in	*Callback*	Function to call on each node
in	*CallbackArg*	Argument to pass to callback

Returns

return code from last callback
0 if the tree is empty

9.32.6.28 int hpss_irbtree_foreach_node (const **hpss_irbtree_node_t** ∗ *Begin,* const **hpss_irbtree_node_t** ∗ *End,* **hpss_irbtree_node_callback_t** *Callback,* void ∗ *CallbackArg*)

Call a function on a range of nodes by forward iteration.

Parameters

in	*Begin*	Node at the beginning of the range (inclusive)

in	*End*	Node at the end of the range (exclusive)
in	*Callback*	Function to call on each node
in	*CallbackArg*	Argument to pass to callback

Returns

return code from last callback
0 if the tree is empty

Note

The range of operation is [Begin, End), i.e. *Begin* is included but *End* is not. Use NULL for *End* if you want a callback on everything up to and including the max element in the tree.

9.32.6.29 int hpss_irbtree_foreach_node_prev (const **hpss_irbtree_node_t** ∗ *RBegin,* const **hpss_irbtree_node_t** ∗ *REnd,* **hpss_irbtree_node_callback_t** *Callback,* void ∗ *CallbackArg*)

Call a function on a range of nodes by reverse iteration.

Parameters

in	*RBegin*	Node at the end of the range (inclusive)
in	*REnd*	Node at the beginning of the range (exclusive)
in	*Callback*	Function to call on each node
in	*CallbackArg*	Argument to pass to callback

Returns

return code from last callback
0 if the tree is empty

Note

The range of operation is (REnd, RBegin], i.e. *RBegin* is included but *REnd* is not. Use NULL for *REnd* if you want a callback on everything up to and including the min element in the tree.

9.32.6.30 void hpss_irbtree_init (**hpss_irbtree_t** ∗ *Tree,* **hpss_irbtree_node_comparator_t** *Comparator*)

Initialize a tree.

Parameters

in	*Tree*	Tree to initialize
in	*Comparator*	Comparator to use for tree

9.32.6.31 **hpss_irbtree_node_t**∗ hpss_irbtree_insert (**hpss_irbtree_t** ∗ *Tree,* **hpss_irbtree_node_t** ∗ *Node*)

Insert a node into a tree.

Parameters

in	*Tree*	Tree to insert into
in	*Node*	Node to insert

Returns

existing node if duplicate found
inserted node if no duplicate found

9.32.6.32 void hpss_irbtree_insert_multi (hpss_irbtree_t ∗ *Tree,* hpss_irbtree_node_t ∗ *Node*)

Insert a (possibly) duplicate node into a tree.

Parameters

in	*Tree*	Tree to insert into
in	*Node*	Node to insert

9.32.6.33 hpss_irbtree_node_t∗ hpss_irbtree_lower_bound (const hpss_irbtree_t ∗ *Tree,* const hpss_irbtree_node_t ∗ *Node*)

Find the lower bound for a key.

This finds the left-most node that is greater-than-or-equal-to the input.

Parameters

in	*Tree*	Tree to search
in	*Node*	Node to compare

Returns

lower-bound node from tree
NULL for no match

9.32.6.34 hpss_irbtree_node_t∗ hpss_irbtree_max (const hpss_irbtree_t ∗ *Tree*)

Get the right-most object in the tree.

This gives you the object that is strictly greater than or equal to all other objects in the tree, according to *Comparator*.

Parameters

in	*Tree*	Tree to search

Returns

the maximum object in the tree
NULL for an empty tree

Note

Starting with the node returned by this, repeatedly calling *hpss_irbtree_node_prev* will iterate through the entire tree in non-ascending node order.

9.32.6.35 hpss_irbtree_node_t∗ hpss_irbtree_min (const hpss_irbtree_t ∗ *Tree*)

Get the left-most object in the tree.

This gives you the object that is strictly less than or equal to all other objects in the tree, according to *Comparator*.

Parameters

in	*Tree*	Tree to search

Returns

the minimum object in the tree
NULL for an empty tree

Note

Starting with the node returned by this, repeatedly calling *hpss_irbtree_node_next* will iterate through the entire tree in non-descending node order.

9.32.6.36 hpss_irbtree_node_t∗ hpss_irbtree_node_next (const hpss_irbtree_node_t ∗ *Node*)

Iterate to the next node in the tree.

Parameters

in	*Node*	Current node

Returns

next node in the tree
NULL for the last node in the tree

9.32.6.37 hpss_irbtree_node_t∗ hpss_irbtree_node_prev (const hpss_irbtree_node_t ∗ *Node*)

Iterate to the previous node in the tree.

Parameters

in	*Node*	Current node

Returns

previous node in the tree
NULL for the first node in the tree

9.32.6.38 void hpss_irbtree_print (const hpss_irbtree_t ∗ *Tree,* hpss_irbtree_node_printer_t *Printer*)

Call a printer function on each node in a tree.

Parameters

in	*Tree*	Tree to print
in	*Printer*	Function to call on each node

9.32.6.39 hpss_irbtree_node_t* hpss_irbtree_remove (hpss_irbtree_t * *Tree,* hpss_irbtree_node_t * *Node,* hpss_irbtree_node_destructor_t *Destructor*)

Remove a node from a tree.

Parameters

in	*Tree*	Tree to remove node from
in	*Node*	Node to remove
in	*Destructor*	Callback to use on node once it has been removed

Returns

next node in the tree

Note

Unlike *hpss_irbtree_find*, you must use a node which is actually a member of *Tree*. This is the only way to be able to disambiguate which node to remove when multiple nodes can be logically equal in the tree. If there is no need to disambiguate and you just want to remove *any* node matching *Node*, then you should do the following:

```
struct myStruct x;
x.ComparisonData = Criteria;

hpss_irbtree_node_t *node = hpss_irbtree_find(&Tree, &x.TreeNode);
if(node != NULL)
  hpss_irbtree_remove(&Tree, node, myDestructor);
```

Similarly, if you want to remove *every* matching node, then you can do the following:

```
struct myStruct x;
x.ComparisonData = Criteria;

hpss_irbtree_node_t *node = hpss_irbtree_find(&Tree, &x.TreeNode);
while(node != NULL)
  node = hpss_irbtree_remove(&Tree, node, myDestructor);
  if(node != NULL && myComparator(node, &x.TreeNode) != 0)
    node = NULL;
}
```

9.32.6.40 size_t hpss_irbtree_size (const hpss_irbtree_t * *Tree*)

Get number of nodes in the tree.

Parameters

in	*Tree*	Tree to get count for

Returns

number of nodes in the tree

9.32.6.41 hpss_irbtree_node_t∗ hpss_irbtree_upper_bound (const hpss_irbtree_t ∗ *Tree,* const hpss_irbtree_node_t ∗ *Node*)

Find the upper bound for a key.

This finds the left-most node that is greater-than the input.

Parameters

`in`	*Tree*	Tree to search
`in`	*Node*	Node to compare

Returns

upper-bound node from tree
NULL for no match

9.33 Hash Interface

Hashing and Checksumming Functions.

Macros

- #define HPSS_CKSUM_AFALG (0x00000004)
- #define HPSS_CKSUM_SSE42 (0x00000002)
- #define HPSS_CKSUM_VPMSUM (0x00000001)

Functions

- int hpss_HashAppend (hpss_hash_t Hash,int Character)

  Append a character to the hash stream.
- int hpss_HashAppendBuf (hpss_hash_t Hash,unsigned char ∗Buffer,unsigned int Length)

  Append a byte buffer to the hash stream.
- int hpss_HashAppendStr (hpss_hash_t Hash,char ∗String)

  Append a null-terminated C string to hash stream.
- void hpss_HashBaseInit (char ∗∗envp)

  Initialize hash hardware flags.
- hpss_hash_t hpss_HashCreate (hpss_hash_type_t Type)

  Create a hash based upon the algorithm type.
- hpss_hash_t hpss_HashDecode (unsigned char ∗Buffer,unsigned int Length)

  Decode a hash context.
- int hpss_HashDelete (hpss_hash_t Hash)

  Free a hashing engine.
- int hpss_HashDigestLength (hpss_hash_type_t Type)

  Returns the number of bytes required to hold the hash digest.
- void hpss_HashDigestToHex (unsigned char ∗digest, int digest_len, unsigned char ∗digest_str)

  Convert a hash digest to hex.
- char ∗ hpss_HashDigestToString (const unsigned char ∗Buffer,unsigned int Length)

  Convert a hash digest buffer into an hex string.
- hpss_hash_t hpss_HashDuplicate (hpss_hash_t Hash)

  Duplicate an existing hash.
- int hpss_HashEncode (hpss_hash_t Hash,unsigned char ∗Buffer,unsigned int Length)

  Encode a hash context.
- int hpss_HashExtract (hpss_hash_t Hash,void ∗Buffer,unsigned int ∗Length,hpss_hash_type_t ∗Type)

  Extract a hashing engine into a buffer.
- unsigned char ∗ hpss_HashFinish (hpss_hash_t Hash,unsigned int ∗Length)

  Finish hashing and return the result.
- int hpss_HashFinishDigest (hpss_hash_t Hash,unsigned char ∗Digest,unsigned int Length)

  Finalize a hash calculation.
- char ∗ hpss_HashFinishHex (hpss_hash_t Hash)

  Return a finished hash as a hex string.
- int hpss_HashGetHWFlags ()

  Return checksum flags.
- hpss_hash_t hpss_HashLoad (void ∗Buffer,unsigned int Length,hpss_hash_type_t Type)

  Load a hash engine from a buffer.
- int hpss_HashReset (hpss_hash_t Hash)

  Reset hash engine in preparation for a new stream.

- int hpss_HashType (hpss_hash_t Hash,hpss_hash_type_t ∗Type)

  Get the hash algorithm type.
- unsigned char ∗ hpss_HexStringToHashDigest (const char ∗HexString,unsigned int ∗Length)

  Convert a hex string into a hash digest buffer.

9.33.1 Detailed Description

Hashing and Checksumming Functions.

9.33.2 Macro Definition Documentation

9.33.2.1 #define HPSS_CKSUM_AFALG (0x00000004)

Support for af-alg

9.33.2.2 #define HPSS_CKSUM_SSE42 (0x00000002)

Support for sse42 (x86)

9.33.2.3 #define HPSS_CKSUM_VPMSUM (0x00000001)

Support for vpmsum (ppc)

9.33.3 Function Documentation

9.33.3.1 int hpss_HashAppend (hpss_hash_t *Hash*, int *Character*)

Append a character to the hash stream.

Parameters

`in`	*Hash*	Hash Engine
`in`	*Character*	Character data to append

Return values

0	Success
HPSS_EFAULT	Invalid hash context
HPSS_EINVAL	Invalid hash type
HPSS_ESYSTEM	Hash engine system failure

9.33.3.2 int hpss_HashAppendBuf (hpss_hash_t *Hash*, unsigned char ∗ *Buffer*, unsigned int *Length*)

Append a byte buffer to the hash stream.

This calls the hash-specific appendbuf function, which handles the operation for this hash.

Parameters

`in`	*Hash*	Hash Engine
`in`	*Buffer*	Data Buffer
`in`	*Length*	Buffer length

Return values

0	Success
HPSS_EFAULT	Invalid hash context or buffer
HPSS_EINVAL	Invalid hash type
HPSS_ESYSTEM	Hash engine system failure

9.33.3.3 int hpss_HashAppendStr (hpss_hash_t *Hash,* char ∗ *String*)

Append a null-terminated C string to hash stream.

Append the contents of a C string to the provided hash stream. The null terminator is not appended to the hash stream.

Parameters

`in`	*Hash*	Hash Engine
`in`	*String*	C String

Return values

0	Success
HPSS_EFAULT	Invalid hash context or string
HPSS_EINVAL	Invalid hash type
HPSS_ESYSTEM	Hash engine system failure

9.33.3.4 void hpss_HashBaseInit (char ∗∗ *envp*)

Initialize hash hardware flags.

Do hardware and OS-specific checks to determine what options are available for accelerated hashing. This is currently only used for crc32c.

Parameters

`in`	*System*	environment

Note

This function should only be run once.
Currently we only support setting one 'support' flag

9.33.3.5 hpss_hash_t hpss_HashCreate (hpss_hash_type_t *Type*)

Create a hash based upon the algorithm type.

This creates a hash engine using the algorithm type supplied.

Parameters

`in`	*Type*	Algorithm Type

Warning

The caller must free the hash engine using hpss_HashDelete in order to reclaim the memory

Returns

An HPSS hash context, or NULL if a hash context cannot be created.

9.33.3.6 hpss_hash_t hpss_HashDecode (unsigned char * *Buffer,* unsigned int *Length*)

Decode a hash context.

This function decode a hash buffer using the contents of a supplied buffer that is suitable for transfer between non-homogeneous machines.

Parameters

`in`	*Buffer*	Buffer used for the decode
`in`	*Length*	Length of encoded hash data

Warning

The caller must free the hash engine using hpss_HashDelete in order to reclaim the memory

Returns

An HPSS hash context, or NULL if a hash context cannot be created.

9.33.3.7 int hpss_HashDelete (hpss_hash_t *Hash*)

Free a hashing engine.

This frees up resources held by the hash engine, then deletes the engine.

Parameters

`in`	*Hash*	Hash Engine

Return values

0	Success
HPSS_EFAULT	Invalid hash context or buffer
HPSS_EINVAL	Invalid hash type

9.33.3.8 int hpss_HashDigestLength (hpss_hash_type_t *Type*)

Returns the number of bytes required to hold the hash digest.

This function returns the number of bytes required to hold the hash digest.

Parameters

in	*Type*	Algorithm Type

Returns

The size need for the digest, or a negative value on failure

Return values

HPSS_EINVAL	Invalid hash type

9.33.3.9 void hpss_HashDigestToHex (unsigned char ∗ *digest,* int *digest_len,* unsigned char ∗ *digest_str*)

Convert a hash digest to hex.

Parameters

in	*digest*	Digest String (binary representation)
in	*digest_len*	Digest String length
in,out	*digest_str*	Digest String (hex representation)

This function takes the binary hash digest value and returns a hex representation of the binary hash digest, in all lower case.

Returns

A lower case hex string buffer.

9.33.3.10 char ∗ hpss_HashDigestToString (const unsigned char ∗ *Buffer,* unsigned int *Length*)

Convert a hash digest buffer into an hex string.

This function converts the contents of the specified hash digest buffer into the equivalent ASCII hex string.

Parameters

in	*Buffer*	Buffer used for the digest binary data
in	*Length*	Number of valid digest bytes

Warning

The caller is responsible for freeing the returned string.

Returns

A hex string, or NULL if the string cannot be allocated

9.33.3.11 hpss_hash_t hpss_HashDuplicate (hpss_hash_t *Hash*)

Duplicate an existing hash.

This duplicates an existing hash, maintaining the state of the original hash.

Parameters

in	*Hash*	Hash Engine

Returns

An identical copy of the hash, or NULL if the hash cannot be duplicated.

9.33.3.12 int hpss_HashEncode (hpss_hash_t *Hash,* unsigned char ∗ *Buffer,* unsigned int *Length*)

Encode a hash context.

This function encodes the existing hash into a buffer that is suitable for transfer between non-homogeneous machines.

Parameters

in	*Hash*	Hash reference
in,out	*Buffer*	Buffer used for the encoded hash
in	*Type*	Algorithm Type

Return values

HPSS_E_NOERROR	Success
HPSS_EFAULT	Invalid hash context or buffer pointer
HPSS_EINVAL	Invalid hash provided

9.33.3.13 int hpss_HashExtract (hpss_hash_t *Hash,* void ∗ *Buffer,* unsigned int ∗ *Length,* hpss_hash_type_t ∗ *Type*)

Extract a hashing engine into a buffer.

This copies the existing hash engine out into a buffer. It also provides the type of the hash engine if needed, and the length of the finished buffer.

Parameters

in	*Hash*	Hash Engine
in,out	*Buffer*	Buffer containing the hash engine
in,out	*Length*	Buffer length
in,out	*Type*	Hash Engine Type

Return values

0	Success
HPSS_EFAULT	Invalid hash context, buffer or length pointer
HPSS_EINVAL	Invalid hash provided

9.33.3.14 unsigned char ∗ hpss_HashFinish (hpss_hash_t *Hash,* unsigned int ∗ *Length*)

Finish hashing and return the result.

Parameters

in	*Hash*	Hash Engine
in,out	*Length*	Digest buffer length

This function finalizes the supplied hash engine and returns an allocated buffer with the binary hash digest value.

Returns

A binary string buffer, or NULL if the operation fails.

Note

This function does not return a printable string; it returns a binary buffer.

9.33.3.15 int hpss_HashFinishDigest (hpss_hash_t *Hash,* unsigned char ∗ *Digest,* unsigned int *Length*)

Finalize a hash calculation.

This function finalizes the supplied hash stream and copies the resulting hash digest into the supplied buffer, returning the digest size.

Parameters

in	*Hash*	Pointer to hash reference
out	*Digest*	Pointer to allocated memory for binary digest
in	*Length*	Size of allocated memory for digest

Returns

The size of the digest, or a negative value on failure

Return values

HPSS_EINVAL	Invalid hash type
HPSS_ENOMEM	Memory allocation failure
HPSS_ESYSTEM	Hash engine system failure

Note

The digest returned is in binary format.

9.33.3.16 char ∗ hpss_HashFinishHex (hpss_hash_t *Hash*)

Return a finished hash as a hex string.

Parameters

in	*Hash*	Hash Engine

This calls hpss_HashFinish, converts the binary hash value to hex. It frees the binary value and returns the hex value. The user should free this string when done.

Returns

Returns a hex string which represents the binary hash value as hex, or NULL if an error occurred.

Note

The string retrieved by this function should be freed by the caller.

9.33.3.17 int hpss_HashGetHWFlags ( )

Return checksum flags.

Returns information about certain hardware-acceleration available for checksumming on this system.

Return values

HPSS_CKSUM_AFALG	Kernel-level checksumming available
HPSS_CKSUM_SSE42	x86 SSE42 hardware acceleration available
HPSS_CKSUM_VPMSUM	ppc hardware acceleration available

9.33.3.18 hpss_hash_t hpss_HashLoad (void ∗ *Buffer,* unsigned int *Length,* hpss_hash_type_t *Type*)

Load a hash engine from a buffer.

This creates a hash engine using the contents of a supplied buffer. This buffer is most likely obtained from hpss_-ExtractHash().

Parameters

`in`	*Buffer*	Data Buffer
`in`	*BufferLen*	Buffer Length
`in`	*type*	Hash Type

Returns

An HPSS hash loaded with the buffer contents, or NULL if the buffer cannot be loaded.

9.33.3.19 int hpss_HashReset (hpss_hash_t *Hash*)

Reset hash engine in preparation for a new stream.

Parameters

`in`	*Hash*	Hash Engine

Return values

0	Success
HPSS_EFAULT	Invalid hash context
HPSS_ESYSTEM	Hash engine system failure

This resets the hash engine to pre-appended state, in preparation to start hashing a new stream. This function is called by hpss_HashCreateXXX, hpss_HashFinish, and hpss_HashFinishHex, so the user will rarely need to call this function explicitly. It also overwrites hash data, which is good for security.

9.33.3.20 int hpss_HashType (hpss_hash_t *Hash,* hpss_hash_type_t ∗ *Type*)

Get the hash algorithm type.

This function returns the type of the hash for the specified hash stream/context.

Parameters

`in`	*Hash*	Hash reference
`out`	*Type*	Pointer to returned algorithm type

Return values

0	Success
HPSS_EINVAL	Invalid hash type
HPSS_EFAULT	Invalid hash or type pointer

9.33.3.21 unsigned char ∗ hpss_HexStringToHashDigest (const char ∗ *HexString,* unsigned int ∗ *Length*)

Convert a hex string into a hash digest buffer.

This function converts an ASCII hex string into the equivalent hash digest buffer.

Parameters

`in`	*HexString*	String of hex digits
`out`	*Length*	Number of valid bytes in return string

Warning

The caller is responsible for freeing the returned string.

Returns

A binary hash digest buffer, or NULL if the string cannot be allocated

9.34 HPSS Config File API

Configuration File Parser APIs.

Classes

- struct hpss_cfg_stanza

  Configuration stanza. More...

Macros

- #define HPSS_CFG_FILE_FMT_ERROR -7001

  File does not appear to be a config file.
- #define HPSS_CFG_FILE_MEM_ERROR -7002

  Memory allocation problem.
- #define HPSS_CFG_FILE_NO_KEY -7007

  No Key Found.
- #define HPSS_CFG_FILE_NO_VALUE -7008

  No Value Found.
- #define HPSS_CFG_FILE_NOT_CMPD -7009

  Not Compound Stanza.
- #define HPSS_CFG_FILE_PARSE_ERR01 -7003

  Parse error - too many levels.
- #define HPSS_CFG_FILE_PARSE_ERR02 -7004

  Syntax error: "{" not preceded by "=".
- #define HPSS_CFG_FILE_PARSE_ERR03 -7005

  Parse error - extraneous closing "}".
- #define HPSS_CFG_FILE_PARSE_ERR04 -7006

  Parse error - missing closing "}".
- #define HPSS_CFG_FILE_SECTION_EMPTY -7010

  Empty Compound Stanza.
- #define HPSS_CFG_FILEOPEN_ERROR -7000

  Unable to open specified file.
- #define HPSS_CFG_MAX_CHARS_PER_LINE 512 /∗ HPSS.conf spec is 128 ∗/
- #define HPSS_CFG_MAX_LEVEL 10
- #define HPSS_CFGOPT_COSSELECT "Select COS"

  HPSS-reserved Level 0 Stanza for COS selection.
- #define HPSS_CFGOPT_HSI "HSI"

  HPSS-reserved Level 0 Stanza for HSI.
- #define HPSS_CFGOPT_HTAR "HTAR"

  HPSS-reserved Level 0 Stanza for HTAR.
- #define HPSS_CFGOPT_LOCALFILE "Local File Path"

  HPSS-reserved Level 0 Stanza for Local File Paths.
- #define HPSS_CFGOPT_MULTINODE "Multinode Table"

  HPSS-reserved Level 0 Stanza for Multinode.
- #define HPSS_CFGOPT_NWOPTIONS "Network Options"

  HPSS-reserved Level 0 Stanza for Network Options.
- #define HPSS_CFGOPT_PFTPC "PFTP Client"

  HPSS-reserved Level 0 Stanza for the PFTP Client.
- #define HPSS_CFGOPT_PFTPCIF "PFTP Client Interfaces"

HPSS-reserved Level 0 Stanza for PFTP Client Interfaces.
- #define HPSS_CFGOPT_PFTPD "PFTP Daemon"

  HPSS-reserved Level 0 Stanza for the PFTP Daemon.
- #define HPSS_CFGOPT_PIPEFILE "Pipe File"

  HPSS-reserved Level 0 Stanza for Pipes.
- #define HPSS_CFGOPT_PSI "PSI"

  HPSS-reserved Level 0 Stanza for PSI.
- #define HPSS_CFGOPT_TA "Transfer Agent"

  HPSS-reserved Level 0 Stanza for Transfer Agent.
- #define HPSS_DEFAULT_STANZA "Default"

  Generic reserved Stanza/SubStanza name(s)
- #define SEARCH_BY_KEY 0

  Search flag for searching by key.
- #define SEARCH_BY_VALUE 1

  Search flag for searching by value.
- #define SEARCH_SUB_LEVELS 1

  Search scope flag for searching all sublevels.
- #define SEARCH_THIS_LEVEL 0

  Search scope flag for searching a stanza level.

Typedefs

- typedef struct hpss_cfg_stanza hpss_cfg_stanza_t

  Configuration stanza.

Enumerations

- enum HPSS_CFG_STANZA_TYPE { CFG_FLAG_TYPE, CFG_SIMPLE_TYPE, CFG_COMPOUND_TYPE }

  Config Extension Stanza Types.

Functions

- hpss_cfg_stanza_t ∗ hpss_CfgFindKey (char const ∗searchStr,hpss_cfg_stanza_t ∗cfgStart,int caseFlag)

  Search for a Key value.
- hpss_cfg_stanza_t ∗ hpss_CfgFindToken (char const ∗searchToken, hpss_cfg_stanza_t ∗cfgStart, int keyOrValue, int subLevels)

  Find a token within a key.
- void hpss_CfgFree (hpss_cfg_stanza_t ∗Head)

  Free Allocated Memory for entire hpss_config parsed structure.
- int hpss_CfgParse (char const ∗cfgPath,hpss_cfg_stanza_t ∗∗Head,int ∗parseErr,int ∗errLine)

  Parse a configuration file.
- hpss_cfg_stanza_t ∗ hpsscfgx_ConfParse (char cfgFile[])

  Parse the configuration file.

9.34.1 Detailed Description

Configuration File Parser APIs. The hpss_CfgXXX functions are intended to provide a generic parser for files such as HPSS.conf, HPSS.COS, etc, which conform to the format defined for the HPSS.conf file, as described in the HPSS.conf.7 man page.

The file consists of up to 10 levels of entries, called "stanzas". Each level may contain zero or more entries, each of which takes on one of the following forms: (terms are used as contained in the HPSS.conf man page, for compatibility):

Comments := A line beginning with either "#" or ";". Comment lines are ignored. Note: comments beginning with '#' can also be contained on a line after any other strings. All characters following the # are ignored.

Flag := a string, which may contain spaces. A "flag" value is not followed by an "=" sign. Multiple flag values may be specified per line.

Simple Stanza: := a string, which may contain spaces, followed by an "=" sign, followed by another string, which may also contain spaces. Newlines may separate parts of a simple stanza.

Compound Stanza: := a string, which may contain spaces, followed by an "=" sign, followed by an opening brace ("{"), followed by zero or more Flags,Simple Stanzas, or Compound Stanzas, and terminated by a closing brace ("}").

Continuation line := a line terminated by a backslash ("\") character. Note that continuation lines for comments are NOT recognized.

Restrictions: -------------

- No Line May contain more than 128 characters.
- Only ten (10) levels are allowed.
- All Compound Structures MUST be terminated by a closing bracket. +++

The API provides functions for:

- parsing a file (hpss_CfgParse)
- searching for a stanza at any level (hpss_CfgSearch)
- deleting a stanza and all of its associated substanzas (hpss_CfgFree)
- traversing a stanza at any level (hpss_CfgTravers)

+++

HPSS reserves a set of Level 0 entries for its own use, which are defined as constants in this file.

9.34.2 Class Documentation

9.34.2.1 struct hpss_cfg_stanza

Configuration stanza.

Stanza structure

Class Members

char ∗	FileName	File name from which the configuration came
unsigned32	Flags32	App-specific flags reserved for application(s)
char ∗	KeyString	Malloced flag string for this entry
char ∗	KeyValue	Malloced NULL or value string for this entry
int	level	Level of this entry (0-HPSS_CFG_MAX_LEVEL)
unsigned32	LineNum	Line number within file where stanza starts
struct hpss_cfg_stanza ∗	next	Linked list of substanzas for this level
size_t	substanzaCount	Number of substanzas for this entry
struct hpss_cfg_stanza ∗	substanzaList	Head of linked list of substanzas for this entry
int	type	Enumerated type for this entry
u_signed64	Val64	App-specific info. Reserved for application(s)

9.34.3 Macro Definition Documentation

9.34.3.1 #define HPSS_CFG_MAX_CHARS_PER_LINE 512 /∗ HPSS.conf spec is 128 ∗/

Maximum characters per line

9.34.3.2 #define HPSS_CFG_MAX_LEVEL 10

Maximum number of levels supported

9.34.4 Typedef Documentation

9.34.4.1 typedef struct hpss_cfg_stanza hpss_cfg_stanza_t

Configuration stanza.

Stanza structure

9.34.5 Enumeration Type Documentation

9.34.5.1 enum HPSS_CFG_STANZA_TYPE

Config Extension Stanza Types.

Enumerator

CFG_FLAG_TYPE	Flag Stanza
CFG_SIMPLE_TYPE	Simple Stanza Type
CFG_COMPOUND_TYPE	Compound Stanza Type

9.34.6 Function Documentation

9.34.6.1 hpss_cfg_stanza_t ∗ hpss_CfgFindKey (char const ∗ *searchStr,* hpss_cfg_stanza_t ∗ *cfgStart,* int *caseFlag*)

Search for a Key value.

Parameters

in	*searchStr*	Key to search for
in	*cfgStart*	Head of level to search
in	*caseFlag*	Whether to be case independent or not

This function is called to search for a Key value for a particular level. If caseFlag is 0, then the search is case-dependent, otherwise, the search is case-independent.

Returns

Function returns NULL if the Key is not found, or a pointer to the stanza for the entry if found.

9.34.6.2 hpss_cfg_stanza_t ∗ hpss_CfgFindToken (char const ∗ *searchToken,* hpss_cfg_stanza_t ∗ *cfgStart,* int *keyOrValue,* int *subLevels*)

Find a token within a key.

Parameters

in	*searchToken*	Token to search for
in	*cfgStart*	Head of level to search
in	*keyOrValue*	Search type: SEARCH_BY_KEY or SEARCH_BY_VALUE
in	*subLevels*	Search scope: SEARCH_THIS_LEVEL or SEARCH_SUB_LEVELS

This function is called to search for a white-space separated token within a key for a particular level.

Returns

Function returns NULL if the token is not found, or a pointer to the stanza for the entry if found.

Note

SEARCH_BY_KEY searches stanza keys SEARCH_BY_VALUE searches stanza values SEARCH_THIS_L-EVEL search this level only SEARCH_SUB_LEVELS search sublevels too
This function calls itself recursively if [subLevels] is non-zero

9.34.6.3 void hpss_CfgFree (hpss_cfg_stanza_t ∗ *Head*)

Free Allocated Memory for entire hpss_config parsed structure.

Parameters

in	*Head*	Head of parsed entries

Frees memory allocated by the Cfg routines. All Key/Value and SubStanza entries associated with the "Head" parameter are freed, and then the "Head" entry itself is freed.

Note

This invalidates the "Head" entry, which must therefore NOT be referenced after a call to this function.

9.34.6.4 int hpss_CfgParse (char const ∗ *cfgPath,* hpss_cfg_stanza_t ∗∗ *Head,* int ∗ *parseErr,* int ∗ *errLine*)

Parse a configuration file.

Parameters

in	*cfgPath*	Pathname of the configuration file to parse
out	*Head*	Returned pointer to head of parsed entries
out	*parseErr*	Pointer to returned parse error code
out	*errLine*	Pointer to returned line number containing error

This function is called to parse a configuration file containing stanzas of the form defined for the HPSS.conf file. See the comments in hpss_config_api.h for details of the file structure.

If the file is parsed successfully, then the head of the linked list of stanzas is returned to the caller in ∗Head. The number of lines in the configuration file is returned in ∗errLine; if a fatal parse error occurs, ∗errLine is also set to the line number (starting at 1) on which the error was detected.

Return values

0	Success
-1	Errors detected

Note

∗parseErr is set to one of the internal parsing error values defined in hpss_config_api.h if the function returns a negative value.
∗errLine is set to the number of lines in the file, or the line number at which an error was detected.
The caller is responsible for calling hpss_CfgFree to free up memory for the list when it is no longer needed.

9.34.6.5 hpss_cfg_stanza_t∗ hpsscfgx_ConfParse (char *cfgFile[]*)

Parse the configuration file.

Parameters

out	*cfgFile*	Returned config file path name

This function is called to parse the configuration file, if it can be found, creating the internal "hpss_CfgEntries" table.

The Configuration File is looked for in a number of places, determined using the following algorithm:

1. The path(s) specified by calling hpsscfgx_ConfSetDirPaths().

2. The path(s) specified by the HPSS_CFG_FILE_PATH environment variable.

3. /usr/local/etc

4. /var/hpss/etc

The name of the Configuration File (Appended to the Path) is HPSS.conf unless the hpss_ConfSetFileName() call is explicitly performed prior to the call to hpsscfgx_ConfParse.

If the hpss_CfgEntries table has already been built, then this call does nothing. It is necessary to free and NULL hpss_CfgEntries to perform this routine more than one time.

Returns

Function returns a pointer to the parsed configuration stanza table on successful exit, or a NULL pointer on failure.

9.35 HPSS Access Control List APIs

APIs for manipulating ACLs using the HACL Library.

Classes

- struct hacl_acl_entry

  Defines an access control list entry. More...

Macros

- #define HACL_M_ALLOW_UNAUTHENTICATED (0x10000000)

  Allow unauthenticatied ACLs.
- #define HACL_M_REQUIRE_PERMS (0x40000000)

  Cause hacl_ConvertstringsToHACL to require permissions.
- #define HACL_M_USE_LOCAL_REALM_SPEC (0x20000000)

  Control explicit realm spec behavior in local ACLs.
- #define HACL_M_USE_MASK_OBJ (0x80000000)

  Print out effective permissions.
- #define HACL_MAX_ENTRY_SIZE

  Maximum HACL entry size.
- #define HACL_MAX_PERMS 8 /∗ rwxcid ∗/

  Maximum string size for permission string.
- #define HACL_MAX_TYPE 24 /∗ foreign_group_delegate ∗/

  Maximum string size for an ACL entry type.
- #define HPSS_EHACL_UNAUTH (-10500)

  Informative error indicating an authentication ACL entry.

Typedefs

- typedef struct hacl_acl_entry hacl_acl_entry_t

  Defines an access control list entry.

Functions

- int hacl_ConvertACLToHACL (ns_ACLConfArray_t ∗ACL_n,hacl_acl_t ∗∗ACL_h)

  Convert ACL to string format.
- int hacl_ConvertHACLPermsToPerms (char ∗HPerms,unsigned char ∗Perms)

  Convert permission string to HPSS format.
- int hacl_ConvertHACLToACL (hacl_acl_t ∗ACL_h,ns_ACLConfArray_t ∗ACL_n)

  Convert ACL to nameserver format.
- int hacl_ConvertHACLToString (hacl_acl_t ∗ACL_h,unsigned Flags,char const ∗EntrySeparator,char ∗∗ACL-_s)

  Convert ACL to a form suitable for printing.
- int hacl_ConvertHACLTypeToType (char ∗HType,unsigned char ∗Type)

  Convert ACL entry type to HPSS format.
- int hacl_ConvertPermsToHACLPerms (unsigned char Perms,char ∗HPerms)

  Convert HPSS permission mask to string.

- int hacl_ConvertStringsToHACL (int NumEntries,char ∗Entries[],char const ∗DefaultRealm,char const ∗WhiteSpaceChars,unsigned Flags,hacl_acl_t ∗∗ACL_h)

  Convert strings to HACL format.
- int hacl_ConvertTypeToHACLType (unsigned char Type,char ∗HType)

  Convert ACL entry type to string.
- int hacl_DeleteHACL (char ∗Path,unsigned32 Options,hacl_acl_t ∗ACL_h)

  Deletes selected ACL entries from the named object.
- int hacl_GetHACL (char ∗Path,unsigned32 Options,hacl_acl_t ∗∗ACL_h)

  Query HPSS to get the ACL for the named object.
- int hacl_SetHACL (char ∗Path,unsigned32 Options,hacl_acl_t ∗ACL_h)

  Replace the ACL of the named object with a new ACL.
- void hacl_SortHACL (hacl_acl_t ∗ACL_h)

  Sort an ACL into canonical order.
- int hacl_UpdateHACL (char ∗Path,unsigned32 Options,hacl_acl_t ∗ACL_h)

  Change the selected ACL entries for the named object.

9.35.1 Detailed Description

APIs for manipulating ACLs using the HACL Library.

9.35.2 Class Documentation

9.35.2.1 struct hacl_acl_entry

Defines an access control list entry.

The hacl_acl_entry_t structure defines a single access control list entry. The hacl_acl_t structure defines a conformant array of ACL entries, with Length defining the number of entries in the array ACLEntry.

Class Members

char	EntryName[HPSS_MAX_PRINCIPAL_NAME]	User, group, or realm name.
char	EntryType[24]	Type of ACL entry.
char	Perms[8]	Character-based permission string. Valid characters are "rwxcid-"
char	RealmName[HPSS_MAX_REALM_NAME]	Realm name which qualifies user and group names.

9.35.3 Macro Definition Documentation

9.35.3.1 #define HACL_MAX_ENTRY_SIZE

Value:

```
(HACL_MAX_TYPE + 1 + HPSS_MAX_REALM_NAME + 1 + \
        HPSS_MAX_PRINCIPAL_NAME + 1 + HACL_MAX_TYPE)
```

Maximum HACL entry size.

The following constant gives the longest ACL entry string. Strings take the form "type:/.../cell/name:rwxcid". The constant includes space for a trailing null.

9.35.3.2 #define HPSS_EHACL_UNAUTH (-10500)

Informative error indicating an authentication ACL entry.

This constant is used by hacl to tell the user when an "authenticated" ACL entry is permitted (only on remove commands).

9.35.4 Typedef Documentation

9.35.4.1 typedef struct hacl_acl_entry hacl_acl_entry_t

Defines an access control list entry.

The hacl_acl_entry_t structure defines a single access control list entry. The hacl_acl_t structure defines a conformant array of ACL entries, with Length defining the number of entries in the array ACLEntry.

9.35.5 Function Documentation

9.35.5.1 int hacl_ConvertACLToHACL (ns_ACLConfArray_t ∗ *ACL_n,* hacl_acl_t ∗∗ *ACL_h*)

Convert ACL to string format.

Parameters

in	*ACL_n*	ACL to convert
out	*ACL_h*	Converted ACL

Function hacl_ConvertACLToHACL converts ACLs from nameserver format into string format.

Return values

0	Success; ACL_h contains converted ACL
EINVAL	Invalid arguments (e.g., unknown ACL entry type)
ENOMEM	Insufficient memory

Note

The sizes of the realm and principal name fields in api_namespec_t and hacl_acl_entry_t must be identical; otherwise data may be lost.
The ACL_n must have an other_obj entry. It will be used to define ACL_h's DefaultRealm field.
This routine tries to fill out every field of every ACL entry with meaningful information, even if the entry doesn't conceptually seem to require that information. In particular, the user_obj entry would not seem to need the RealmName to be filled out since the name is implicitly defined by the object's owner. However, this routine fills out the RealmName anyhow so as to give a context which other routines can use to interpret the other ACL entries.
Some obscure entry types (eg., user_obj_delegate) have been treated rather cavalierly. It's not clear to me whether these entries will contain meaningful information to translate, so have chosen to not translate them.
If this routine succeeds, caller must free() ACL_h when done with it.

9.35.5.2 int hacl_ConvertHACLPermsToPerms (char ∗ *HPerms,* unsigned char ∗ *Perms*)

Convert permission string to HPSS format.

Parameters

in	*HPerms*	Permissions to Convert
out	*Perms*	Converted Permissions

Function hacl_ConvertHACLPermsToPerms converts a permission string to HPSS nameserver format. The string can contain any of the characters "rwxcidRWXCID-" in any order. Hyphens are ignored. Other characters are used to control which bit positions get set in Perms.

Return values

0	Success
EINVAL	HPerms contains one or more invalid characters

Note

HPerms must be a null-terminated string.

9.35.5.3 int hacl_ConvertHACLToACL (hacl_acl_t ∗ *ACL_h,* ns_ACLConfArray_t ∗ *ACL_n*)

Convert ACL to nameserver format.

Parameters

in	*ACL_h*	ACL to Convert
out	*ACL_n*	Converted ACL

Return values

0	Success; ACL_n contains converted ACL
EINVAL	Invalid arguments
ENOMEM	Insufficient memory

Note

If this routine succeeds, caller must free() ACL_n when done with it.
In order to keep the nameserver from accidentally changing the owner of an object as a side effect of the ACL change, this routine inserts "ignore me" values into some ACL entries (e.g., user_obj).

9.35.5.4 int hacl_ConvertHACLToString (hacl_acl_t ∗ *ACL_h,* unsigned *Flags,* char const ∗ *EntrySeparator,* char ∗∗ *ACL_s*)

Convert ACL to a form suitable for printing.

Parameters

in	*ACL_h*	ACL to Convert
in	*Flags*	Flags
in	*EntrySeparator*	Entry Separator
out	*ACL_s*	Converted ACL

Return values

0	Success

EINVAL	Invalid arguments
ENOMEM	Insufficient memory

Note

ACL_h->DefaultRealm must be set up correctly if local entries are to have the realm information displayed correctly.
If the HACL_M_USE_MASK_OBJ bit is set in Flags, the routine will compute the effective permissions, taking into account the mask object. Otherwise the permission bits of each ACL entry are shown "as is".
In typical use, one expects all "user" ACL entries to name people that reside in the realm where the object owner resides. In this case, the user entries will be displayed as "user:name:perms". However, in some exceptional cases, a user entry may refer to someone in a different realm. If the HACL_M_USE_LOCAL_R-EALM bit is not set in Flags, the ACL entry will still be printed in the form "user:name:perms". This ensures that all user entries are printed in the same format that dcecp and acl_edit use. However, the resulting output maybe be misleading. On the other hand, if the HACL_M_USE_LOCAL_REALM is set in Flags, then the routine will try to be intelligent about what is displayed. If the user entry refers to the object owner's realm, the routine will still display the entry as "user:name:perms", but otherwise will display it as "user:/realm/name-:perms". Similar comments apply to other local entries such as group entries.
This function uses a two-pass loop to compute the string. In the first pass, the length of the string is computed. In the second pass, memory is allocated for the string and then data is copied to the string. Macro COPY is used to implement this.

9.35.5.5 int hacl_ConvertHACLTypeToType (char ∗ *HType,* unsigned char ∗ *Type*)

Convert ACL entry type to HPSS format.

Parameters

`in`	*HType*	Type to Convert
`out`	*Type*	Converted Type

Function hacl_ConvertHACLTypeToType converts an ACL entry type string to HPSS nameserver format. The string can contain any of the legal DCE ACL types except "extended". Upper and lower case letters are accepted.

Return values

0	Success
EINVAL	HType was not a recognized ACL type

9.35.5.6 int hacl_ConvertPermsToHACLPerms (unsigned char *Perms,* char ∗ *HPerms*)

Convert HPSS permission mask to string.

Parameters

`in`	*Perms*	Permissions to Convert
`out`	*HPerms*	Converted Permissions

Function hacl_ConvertPermsToHACLPerms converts permission masks from HPSS format to string format. The string is always printed using lower case letters, with hyphens used to show permissions that are not granted.

Return values

0	Success

EINVAL	HPerms is NULL

Note

Caller supplies storage for HPerms sufficient to hold at least HACL_MAX_PERMS characters.

9.35.5.7 int hacl_ConvertStringsToHACL (int *NumEntries,* char ∗ *Entries[ ],* char const ∗ *DefaultRealm,* char const ∗ *WhiteSpaceChars,* unsigned *Flags,* hacl_acl_t ∗∗ *ACL_h*)

Convert strings to HACL format.

Parameters

in	*NumEntries*	Number of array elements
in	*Entries*	Array of ACL strings
in	*DefaultRealm*	Default Realm
in	*WhiteSpace-Chars*	ACL entry separators
in	*Flags*	Flags
out	*ACL_h*	Access Control List

Function hacl_ConvertStringsToHACL converts one or more strings representing acl entries into a single access control list.

Return values

0	The function succeeded
EINVAL	An ACL entry was not in the right format
ENAMETOOLONG	Default realm name is too long
ENOMEM	Insufficient memory

Note

If this routine succeeds, caller must free() ACL_h when done with it. The routine can be used in several ways. On the one hand, if NumEntries equals 1, then only Entries[0] needs to be provided. It will be a string containing several ACL entries, each one separated by one or more characters specified by WhiteSpace-Chars. On the other hand, NumEntries can be greater than 1, in which case the Entries array must have that many elements. Each entry in the array can then define a single ACL entry. These approaches can be mixed, so that several Entries can be provided, each of which defines several ACL entries. Regardless of which approach is used, all of the ACL entries will be merged into a single access control list which will be returned in ACL_h. Note that in general, there is no guarantee that (∗ACL_h)->NumEntries == NumEntries.
White space can be anything, including commas, semicolons, spaces, tabs, quotes and/or newlines.
See the comments in ParseEntry for details on how this routine deals with entries that don't require specific values for the EntryName and/or RealmName fields. (It supplies blanks.)

9.35.5.8 int hacl_ConvertTypeToHACLType (unsigned char *Type,* char ∗ *HType*)

Convert ACL entry type to string.

Parameters

in	*Type*	ACL entry type to convert

out	*HType*	Converted Type

Function hacl_ConvertTypeToHACLType converts an ACL entry type from nameserver format into a string suitable for printing. The input is a value (e.g. ACL_USER_OBJ), and the output is a string (e.g., "user_obj").

Return values

0	Success
EINVAL	Type argument is invalid

Note

Caller supplies storage for HType sufficient to hold at least HACL_MAX_TYPE characters.

9.35.5.9 int hacl_DeleteHACL (char ∗ *Path,* unsigned32 *Options,* hacl_acl_t ∗ *ACL_h*)

Deletes selected ACL entries from the named object.

Parameters

in	*Path*	Path of Object
in	*Options*	Processing Options
in	*ACL_h*	ACL to be removed

Return values

0	Success
Other	Error code indicating the nature of the error

9.35.5.10 int hacl_GetHACL (char ∗ *Path,* unsigned32 *Options,* hacl_acl_t ∗∗ *ACL_h*)

Query HPSS to get the ACL for the named object.

Parameters

in	*Path*	Path of Object
in	*Options*	Processing Options
in	*ACL_h*	Object ACL

Return values

0	Success
EINVAL	Invalid arguments

9.35.5.11 int hacl_SetHACL (char ∗ *Path,* unsigned32 *Options,* hacl_acl_t ∗ *ACL_h*)

Replace the ACL of the named object with a new ACL.

Parameters

in	*Path*	Path of Object

in	*Options*	Processing Options
in	*ACL_h*	New ACL

Return values

0	Success
Other	Error indicating the nature of the error

9.35.5.12 void hacl_SortHACL (hacl_acl_t * *ACL_h*)

Sort an ACL into canonical order.

Parameters

in,out	*ACL_h*	ACL to be sorted

Function hacl_SortHACL sorts an ACL into canonical order. The order is defined so that ACL entries are printed in the order used by acl_edit and dcecp. For example, this puts user type entries before group type entries, and all delegate entries after non-delegate entries. Presumably this order is also the same order that DCE uses to decide whether a principal has access to an object.

Note

Entry types must be specified using lowercase letters.

9.35.5.13 int hacl_UpdateHACL (char * *Path,* unsigned32 *Options,* hacl_acl_t * *ACL_h*)

Change the selected ACL entries for the named object.

Parameters

in	*Path*	Path of Object
in	*Options*	Processing Options
in	*ACL_h*	ACL to be updated

Return values

0	Success
Other	Error indicating the nature of the failure

9.36 Gatekeeper Site Library

Prototypes for HPSS Gatekeeper site interfaces.

Classes

- struct gk_EntryInfo

  Gatekeeper Request Information Entry. More...
- struct gk_ReqInfo

  Gatekeeper RTM info. More...

Typedefs

- typedef struct gk_EntryInfo gk_EntryInfo_t

  Gatekeeper Request Information Entry.
- typedef struct gk_ReqInfo gk_ReqInfo_t

  Gatekeeper RTM info.

Functions

- signed32 gk_site_Close (hpss_uuid_t ControlNo)

  Called when an HPSS file is being opened by an Authorized Caller.
- signed32 gk_site_CreateComplete (hpss_uuid_t ControlNo)

  Called when an HPSS file create has been completed.
- void gk_site_CreateStats (gk_EntryInfo_t EntryInfo)

  Called when an HPSS file is being created by an Authorized Caller.
- signed32 gk_site_GetMonitorTypes (u_signed64 ∗MonitorTypeBitsP)

  Called to get the request types being monitored.
- signed32 gk_site_Init (char ∗SitePolicyPathNameP)

  Called to initialize the Site Interface.
- signed32 gk_site_Open (gk_EntryInfo_t EntryInfo, unsigned32 ∗WaitTimeP)

  Called when an HPSS file is being opened.
- void gk_site_OpenStats (gk_EntryInfo_t EntryInfo)

  Called when an HPSS file is being opened by an Authorized Caller.
- signed32 gk_site_Stage (gk_EntryInfo_t EntryInfo, unsigned32 ∗WaitTimeP)

  Called when an HPSS file is being staged.
- signed32 gk_site_StageComplete (hpss_uuid_t ControlNo)

  Called when an HPSS file stage has completed.
- void gk_site_StageStats (gk_EntryInfo_t EntryInfo)

  Called when an HPSS file is being staged by an Authorized Caller. Authorzied Caller.

9.36.1 Detailed Description

Prototypes for HPSS Gatekeeper site interfaces. This file contains full ANSI C prototypes of all the Site Interface procedures used in the HPSS Gatekeeper Server.

9.36.2 Class Documentation

9.36.2.1 struct gk_EntryInfo

Gatekeeper Request Information Entry.

Defines the the Gatekeeper Server's internal entry structure which is passed to the site routines.

Class Members

unsigned32	AccountId	Account Id.
unsigned32	AuthorizedCaller	Whether the user is an authorized caller.
hpssoid_t	BitFileId	Bitfile Id.
hpss_uuid_t	ClientConnectId	End Client's connection id.
hpss_uuid_t	ConnectionId	Core Server connection id.
hpss_uuid_t	ControlNo	Unique entry control number.
unsigned32	Cos	Class of Service Id.
unsigned32	GroupId	User's active group Id.
hpss_sockaddr-_t	HostAddr	Originating host.
gk_Open_t	OpenSpecificInfo	Additional info specific to an open request.
unsigned32	RealmId	User's Realm Id.
hpss_reqid_t	RequestId	User's request id.
gk_Request-Type_t	RequestType	Request Type.
gk_ReqInfo_t *	RTMReqInfoP	RTM Request Information. Real-time monitoring request information for retry logic
gk_Stage_t	StageSpecific-Info	Additional info specific to a stage request.
timestamp_sec-_t	StartTime	Timestamp when the entry was created.
gk_State_t	State	Current state of the entry.
unsigned32	UserId	User's Id.
unsigned32	WaitTime	WaitTime for this entry.

9.36.2.2 struct gk_ReqInfo

Gatekeeper RTM info.

The following structure contains information about an RTM request entry.

Class Members

rtm_request_-code_t	RequestCode	Request Code
hpss_reqid_t	RequestId	Request Id
rtm_req_entry_t *	RTMReqEntryP	RTM Request Entry
rtm_wait_entry_t *	RTMWaitEntryP	Request Wait Entry
rtm_-serverspecific_t	ServerSpecific	Server specific RTM info

9.36.3 Typedef Documentation

9.36.3.1 typedef struct **gk_EntryInfo gk_EntryInfo_t**

Gatekeeper Request Information Entry.

Defines the the Gatekeeper Server's internal entry structure which is passed to the site routines.

9.36.3.2 typedef struct **gk_ReqInfo gk_ReqInfo_t**

Gatekeeper RTM info.

The following structure contains information about an RTM request entry.

9.36.4 Function Documentation

9.36.4.1 **signed32** gk_site_Close (hpss_uuid_t *ControlNo*)

Called when an HPSS file is being opened by an Authorized Caller.

Parameters

`in`	*ControlNo*	Closed file's unique id

This is an internal, site defined-and-implemented function that is called by the Gatekeeper Server whenever a file is closed. This function is not an RPC. It is a call to a procedure in a shared library.

This function is written by the customer site. It will be called by gk_Close while processing a close of an HPSS file.

Return values

0	Success

Warning

Note this routine is only called when monitoring GK_MONITOR_OPEN

See Also

gk_site_CreateComplete

9.36.4.2 **signed32** gk_site_CreateComplete (hpss_uuid_t *ControlNo*)

Called when an HPSS file create has been completed.

Parameters

`in`	*ControlNo*	Created file's unique id

This is an internal, site defined-and-implemented function that is called by the Gatekeeper Server whenever a file create has completed. This function is not an RPC. It is a call to a procedure in a shared library.

This function is written by the customer site. It will be called by gk_CreateComplete while processing a file create completion of an HPSS file.

Return values

0	Success

Warning

This routine is only called when monitoring GK_MONITOR_CREATE

Note

Please see gk_site_Close

A pseudocode example:

```
  Lock(s)
       Find ControlNo in request cache.
       If not found, unlock, log and return HPSS_ENOENT.
       If this was a request that we returned HPSS_E_NOERROR for the
         gk_site_Create, then Decrement stats accordingly
       else If this was a request that we returned HPSS_ERETRY for the
         gk_site_Create, then Decrement stats accordingly
       Remove from request cache.
       Unlock(s).
*
```

9.36.4.3 void gk_site_CreateStats (gk_EntryInfo_t *EntryInfo*)

Called when an HPSS file is being created by an Authorized Caller.

Parameters

in	*EntryInfo*	Information about the create

This is an internal, site defined-and-implemented function that is called asynchronously by the Gatekeeper Server whenever an authorized caller is creating a file. It is only called when authorized caller requests and create requests are being monitored. This function is not an RPC. It is a call to a procedure in a shared library.

This function is written by the customer site. It is called asynchronously by gk_Create whenever an HPSS file create occurs from an authorized caller when both create requests and authorized caller requests are monitored.

Warning

This routine is only called when monitoring GK_MONITOR_AUTHORIZED_CALLER and GK_MONITOR_C-REATE.

```
   Log entrance of this routine to the Site Policy Logfile using the
        RequestId passed in the EntryInfo.
   If we're not monitoring for creates, then
      Log an ALARM Error to the Site Policy Logfile.
      Return.

   If there is a no Site Policy file
      Log an Error to the Site Policy Logfile.
      Return.

   Lock the Request Cache.
   Find the Request in the Request Cache.
   If not found, then
       Create a new Request Entry and add it to the Request Cache.

   If there exists a "Maximum Number of Creates Per Host Policy", then
      Note: Translating socket address into hostnames is expensive, so
```

```
            maintain a cache to translate HostAddrs to HostNames.
      Lock the HostAddrToHostName Cache.
      Lookup the EntryInfo.HostAddr in the HostAddrToHostName Cache.
      If it doesn't exist, then
         hpss_GetHostByAddr
         If Error returned by multithreaded safe get host by addr lookup, then
             Log an error to the Site Policy Logfile.
             Delete the Request from the Request Cache.
             Unlock all locks.
             Return.
         Create, initialize and add a new HostAddrToHostName cache entry.
      Save away the HostName.
      Unlock the HostAddrToHostName Cache.

      Lookup the HostName in the Host Entry Cache.
      Note: Assume the Host Entry Cache is sharing the Request Cache Lock.
      If not found, then
         Create a new Host Entry and add it to the Host Cache.

   If there exists a "Maximum Number of Creates Per User Policy", then

      Lookup the UserId/RealmId pair in the User Entry Cache.
      Note: The UserId and RealmId together will uniquely identify a
            particular user in a federated name space (cross realm)
            environment.
      Note: The User Entry Cache is sharing the Request Cache Lock.
      If not found, then
         Create a new User Entry and add it to the User Cache.

   Mark the Request Entry's state as GOOD.
   Unlock the Request Cache.
   Log exit of this routine to the Site Policy Logfile.
   Return.
*
```

9.36.4.4 signed32 gk_site_GetMonitorTypes (u_signed64 ∗ *MonitorTypeBitsP*)

Called to get the request types being monitored.

Parameters

in	*MonitorTypeBitsP*	Bitvector of items being monitored

This interface will lookup the local site policies to determine which request types are being monitored. It will then return a bitvector of items being monitored. The Bitfile Server will call gk_GetMonitorTypes when (re)connecting to the Gatekeeper to learn which request types are being monitored so that it can call the Gatekeeper appropriately. The Gatekeeper will call gk_site_GetMonitorTypes. Currently the following request types can be monitored: authorized caller, create, open, and stage.

It is important that the Site Interfaces return a status in a timely fashion. Create, open, and stage requests from MPS and other authorized callers are timing sensitive, thus the Site Interfaces won't be permitted to delay or stop these requests, however the Site Interfaces may choose to be involved in keeping statistics on these requests by monitoring requests from authorized callers.

This is an internal, site defined and implemented function that is called by gk_GetMonitorTypes. This function is not an RPC. It is a call to a procedure in a shared library.

MonitorTypeBitsP A bit vector (0 origin array of bits) in which the appropriate bit is set (on) for each request type that is currently being monitored. Request types bit positions are: GK_MONITOR_AUTHORIZED_CALLER = 0 GK_MONITOR_CREATE = 1 GK_MONITOR_OPEN = 2 GK_MONITOR_STAGE = 3

Return values

0	Success

Note

Example Use: The site could have a policy file (SitePolicyPathNameP) where the types of requests being monitored are listed. When the Gatekeeper Server initializes, it'll call gk_site_Init() which will then read the SitePolicyPathNameP and store the request monitor types in a Global variable. This routine will then just merely copy the Global variable into ∗MonitorTypeBitsP.

9.36.4.5 signed32 gk_site_Init (char ∗ *SitePolicyPathNameP*)

Called to initialize the Site Interface.

Parameters

`in`	*SitePolicyPath-NameP*	Path name of the file where the site policy is stored

This is an internal, site defined and implemented function that is called when the Gatekeeper is (re)initialized. This function is used to do whatever initialization is needed by the site module. This function is not an RPC. It is a call to a procedure in a shared library.

Return values

0	Success

Note

Example Use: Initialize any global variables. (Save the SitePolicyPathNameP) Read the SitePolicyPathName-P - setup Globals/etc accordingly. (eg To the right of the string "MonitorTypes = " will be placed a list of strings like "CREATE" "OPEN" which will indicate the request types being monitored. This line will be parsed and then the appropriate bits in the global G_MonitorTypeBits will be set. So, if the policy file contained the line: MonitorTypes = CREATE OPEN Then G_MonitorTypeBits will be setup as: G_MonitorTypeBits = cast64m(0); G_MonitorTypeBits = orbit64m(G_MonitorTypeBits, GK_MONITOR_CREATE); G_MonitorType-Bits = orbit64m(G_MonitorTypeBits, GK_MONITOR_OPEN); Then the G_MonitorTypeBits may be returned in calls to gk_site_GetMonitorTypes.) Close the SitePolicyPathNameP Set up logging to a LogFile specified in SitePolicyPathNameP (optional) Initialize mutexes, structures, etc. (eg May want to initialize cache(s) associ-ated with requests.)

9.36.4.6 signed32 gk_site_Open (gk_EntryInfo_t *EntryInfo*, unsigned32 ∗ *WaitTimeP*)

Called when an HPSS file is being opened.

Parameters

`in`	*EntryInfo*	Information about the open
`out`	*WaitTimeP*	Seconds before retrying

This is an internal, site defined-and-implemented function that is called by the Gatekeeper Server whenever a file is opened. This function is not an RPC. It is a call to a procedure in a shared library.

This function is written by the customer site. It will be called by gk_Open while processing an open of an HPSS file.

Return values

0	Success

Warning

This routine is only called when monitoring GK_MONITOR_OPEN.

See Also

gk_site_Create

9.36.4.7 void gk_site_OpenStats (gk_EntryInfo_t *EntryInfo*)

Called when an HPSS file is being opened by an Authorized Caller.

Parameters

`in`	*EntryInfo*	Information about the open

This is an internal, site defined-and-implemented function that is called asynchronously by the Gatekeeper Server whenever an authorized caller is opening a file. It is only called when authorized caller requests and open requests are being monitored. This function is not an RPC. It is a call to a procedure in a shared library.

This function is written by the customer site. It is called asynchronously by gk_Open whenever an HPSS file open occurs from an authorized caller when both open requests and authorized caller requests are monitored.

Warning

This routine is only called when monitoring GK_MONITOR_AUTHORIZED_CALLER and GK_MONITOR_O-PEN.

See Also

gk_site_CreateStats

9.36.4.8 signed32 gk_site_Stage (gk_EntryInfo_t *EntryInfo*, unsigned32 * *WaitTimeP*)

Called when an HPSS file is being staged.

Parameters

`in`	*EntryInfo*	Information about the stage
`out`	*WaitTimeP*	Secounds before retrying

This is an internal, site defined-and-implemented function that is called by the Gatekeeper Server whenever a file is being staged. This function is not an RPC. It is a call to a procedure in a shared library.

This function is written by the customer site. It will be called by gk_Stage while processing a stage of an HPSS file.

Return values

0	Success

Note

This routine only called when monitoring GK_MONITOR_STAGE.

See Also

gk_site_Create

9.36.4.9 signed32 gk_site_StageComplete (hpss_uuid_t *ControlNo*)

Called when an HPSS file stage has completed.

Parameters

in	*ControlNo*	Staged file's unique id

This is an internal, site defined-and-implemented function that is called by the Gatekeeper Server whenever a file stage has completed. This function is not an RPC. It is a call to a procedure in a shared library.

This function is written by the customer site. It will be called by gk_StageComplete while processing a file stage completion of an HPSS file.

Return values

0	Success

Note

This routine is only called when monitoring GK_MONITOR_STAGE

See Also

gk_site_CreateComplete

9.36.4.10 void gk_site_StageStats (gk_EntryInfo_t *EntryInfo*)

Called when an HPSS file is being staged by an Authorized Caller. Authorzied Caller.

Parameters

in	*EntryInfo*	Information about the stage

This is an internal, site defined-and-implemented function that is called asynchronously by the Gatekeeper Server whenever an authorized caller is staging a file. It is only called when authorized caller requests and stage requests are being monitored. This function is not an RPC. It is a call to a procedure in a shared library.

This function is written by the customer site. It is called asynchronously by gk_Stage whenever an HPSS file stage occurs from an authorized caller when both stage requests and authorized caller requests are monitored.

Note

This routine is only called when monitoring GK_MONITOR_AUTHORIZED_CALLER and GK_MONITOR_S-TAGE.

See Also

gk_site_CreateStats

9.37 Account Validation Site Library

Account Validation Site Library Prototypes.

Functions

- signed32 av_site_AcctIdxToName (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,unsigned32 Gid,unsigned32 Flags,acct_rec_t Acct,char ∗AcctName,unsigned32 ∗OkToCache)

  Convert the supplied account index into an account name.
- signed32 av_site_AcctNameToIdx (hpss_connect_handle_t Binding, hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,unsigned32 Gid,unsigned32 Flags,char ∗InAcctName,char ∗OutAcctName,acct_-rec_t ∗Acct,unsigned32 ∗OkToCache)

  Convert account name to account index.
- signed32 av_site_Initialize (acct_config_t ∗AcctPolicy)

  Initialize site code.
- signed32 av_site_Shutdown (void)

  Shutdown site code.
- signed32 av_site_ValidateAccount (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,acct_rec_t Acct,unsigned32 Flags,acct_rec_t ∗ParentAcct,acct_rec_t ∗Out-Acct,unsigned32 ∗OkToCache)

  Validate a user's account index.
- signed32 av_site_ValidateChacct (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 UserRealmId,unsigned32 UserUid,unsigned32 FileRealmId,unsigned32 FileUid,unsigned32 FileGid,acct_-rec_t OldAcct,acct_rec_t NewAcct,acct_rec_t ∗OutAcct,unsigned32 ∗OkToCache)

  Determine account change index.
- signed32 av_site_ValidateChown (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 Old-RealmId,unsigned32 OldUid,unsigned32 OldGid,acct_rec_t OldAcct,unsigned32 NewRealmId,unsigned32 NewUid,unsigned32 NewGid,acct_rec_t SessionAcct,acct_rec_t ∗OutAcct,unsigned32 ∗OkToCache)

  Determine file ownership index.
- signed32 av_site_ValidateCreate (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,unsigned32 Gid,acct_rec_t Acct,acct_rec_t ParentAcct,hpss_Attrs_t ∗Parent-Attrs,acct_rec_t ∗OutAcct,unsigned32 ∗OkToCache)

  Determine new file/directory account index.

9.37.1 Detailed Description

Account Validation Site Library Prototypes.

9.37.2 Function Documentation

9.37.2.1 signed32 av_site_AcctIdxToName (hpss_connect_handle_t *Binding,* hpss_reqid_t *RequestId,* unsigned32 *RealmId,* unsigned32 *Uid,* unsigned32 *Gid,* unsigned32 *Flags,* acct_rec_t *Acct,* char ∗ *AcctName,* unsigned32 ∗ *OkToCache*)

Convert the supplied account index into an account name.

Parameters

in	*Binding*	RPC Binding Info
in	*RequestId*	Id of this request
in	*RealmId*	User's Realm Id
in	*Uid*	User id
in	*Gid*	User's Group Id
in	*Flags*	Zero or more of the following bit flags: • ACCT_FLAGS_VALID_INDEX - Make sure this user is allowed to use this account. If not, return HPSS_EPERM
in	*Acct*	Account Index to Convert
out	*AcctName*	Name of the Account
out	*OkToCache*	TRUE if this information may be cached by the client.

This site customizable function converts the supplied account index for the given site into an account name. Implementation is up to the individual site. If this routine is not implemented by a site, it should return HPSS_EUSEDEFAULT to tell account validation to perform the default behavior.

Return values

HPSS_EUSEDEFAULT	If the routine is not implemented by the site, use the corresponding default routine
0	Success
HPSS_EBUSY	The server is busy. The reqest should be tried again later.
HPSS_ENOENT	That account index does not exist
HPSS_EPERM	User is not a member of that account
HPSS_ESYSTEM	This routine received an internal eror; usually a site customization problem.
<0	Some other error.

See Also

av_default_AcctIdxToName

A pseudocode example:

```
      av_AcctIdxToName() {
        if site style accounting {
                if Acct == ACCT_REC_DEFAULT
                        lookup user default acct;
                else lookup account Acct;
        if (not found) {
                if Flags & ACCT_FLAGS_VALID_INDEX
                        error = HPSS_EPERM;
                else error = HPSS_ENOENT;
         }
         else if (unix style accounting)
                 if (Acct == ACCT_REC_DEFAULT) Acct = Uid;
                 if (Acct != Uid && Flags & ACCT_FLAGS_VALID_INDEX)
                         error = HPSS_EPERM;
                 else {
                         convert Realm,Acct into name
                         if not found && Flags & ACCT_FLAGS_VALID_INDEX
                                 error = HPSS_EPERM;
                 }
         }
    return error;
}
```

Note

The OkToCache should normally be TRUE. If a site customizes this routine it should only set this argument to FALSE if the returned informatio can change over time and should not be cached by the client.

9.37.2.2 signed32 av_site_AcctNameToIdx (hpss_connect_handle_t *Binding*, hpss_reqid_t *RequestId*, unsigned32 *RealmId*, unsigned32 *Uid*, unsigned32 *Gid*, unsigned32 *Flags*, char * *InAcctName*, char * *OutAcctName*, acct_rec_t * *Acct*, unsigned32 * *OkToCache*)

Convert account name to account index.

Parameters

in	*Binding*	RPC Binding Information for this Request
in	*RequestId*	Id of this Request
in	*RealmId*	Realm Id of the User
in	*Uid*	User's Id (unix uid)
in	*Gid*	Group Id of the user/file (unix gid)
in	*Flags*	Zero or more of the following bit flags: • ACCT_FLAGS_VALID_INDEX - Make sure this user is allowed to use this account.
in	*InAcctName*	Optional Account Name to convert
out	*OutAcctName*	Returned name of the Account
out	*Acct*	Returned account index for this account
out	*OkToCache*	TRUE if this information may be cached by the client.

This site customizable function converts the supplied account name for the given site into an account index. Implementation is up to the individual site. If this routine is not implemented by a site, it should return HPSS_EUSEDEFAULT to tell account validation to perform the default behavior.

Return values

HPSS_EUSEDEFAULT	If this routine is not implemented by the site, use the corresponding default routine.
0	Success
HPSS_EBUSY	The server is busy. The request should be retried later.
HPSS_ENOENT	That account index does not exist
HPSS_EPERM	User is not a member of that account.
HPSS_ESYSTEM	This routine received an internal error. Usually a site customization problem.
<0	Some other error

See Also

av_default_AcctNameToIdx

Note

The OkToCache should normally be TRUE. If a site customizes this routine it should only set this argument to FALSE if the returned informatio can change over time and should not be cached by the client.

```
av_AcctNameToIdx() {
       lookup_default = (InAcctName is NULL or empty);
       if site style accounting {
               if lookup_default
                       read default account for this user from metadata
                       if (not found and requiring default account)
                               return HPSS_EPERM;
               else if Flags & ACCT_FLAGS_VALID_INDEX
                       read this user's account by name
                else
                        read account metadata by name
        } else if unix style accounting {
                if (lookup_default)
                      Acct = Uid;
                else {
                        try to convert AcctName to number
                        if successful, *Acct = number;
                }
        }

        if we set *Acct above {
                error = lookup user from uid
                if not found and require default account
                        error = HPSS_EPERM;
        } else {
                error = lookup user from account name
                if (not found and Flags & ACCT_FLAGS_VALID_INDEX
                        error = HPSS_EPERM;
        }
        return error;
}
```

9.37.2.3 signed32 av_site_Initialize (acct_config_t * *AcctPolicy*)

Initialize site code.

This function allows the site to perform any needed initialization. For example, you could initialize globals, or allocate memory, among other things, here.

Parameters

in	*AcctPolicy*	Account Policy metadata record

Return values

0	Success
HPSS_ESYSTEM	Site code cannot initialize properly
HPSS_ENOMEM	Out of memory.
<0	Some other fatal error

Note

If you return an error here, the Core Server will abort during initialization.

```
av_site_Initialize() {
  allocate space for local site policy information;
       read in local site policy information;
}
```

9.37.2.4 signed32 av_site_Shutdown (void)

Shutdown site code.

This function allows the site to perform any needed shutdown specific code. This routine is called when HPSS performs a slow shutdown for the server (i.e. it is not guarenteed to be called).

Return values

0	Success
HPSS_EINVAL	The APIs have not been initialized so there is nothing to shut down.
<0	Some other error that will be logged but will not impede the shutdown

```
av_site_Shutdown() {
  deallocate space for local site policy information;
}
```

9.37.2.5 signed32 av_site_ValidateAccount (hpss_connect_handle_t *Binding*, hpss_reqid_t *RequestId*, unsigned32 *RealmId*, unsigned32 *Uid*, acct_rec_t *Acct*, unsigned32 *Flags*, acct_rec_t * *ParentAcct*, acct_rec_t * *OutAcct*, unsigned32 * *OkToCache*)

Validate a user's account index.

This site customizable function is called by the Core Server to verify that a user is permitted to use the specified account index. Implementation is up to the individual site. If this routine is not implemented by a site, it should return HPSS_EUSEDEFAULT to tell account validation to perform the default behavior.

Parameters

in	*Binding*	RPC binding information
in	*RequestId*	Id of this Request
in	*RealmId*	User's Realm Id
in	*Uid*	User's Id (unix uid)
in	*Acct*	Account Index to validate
in	*Flags*	Zero or more of the following bit flags: • ACCT_VALIDATE_FLAGS_CREATING - A file or directory being created.
in	*ParentAcct*	Account index of parent directory
out	*OutAcct*	Account index to use
out	*OkToCache*	TRUE if this information may be cached by the client.

Return values

HPSS_EUSEDEFAULT	If the routine is not implemented by the site, use the corresponding default routine
0	Success
HPSS_EBUSY	The server is busy. The reqest should be tried again later.
HPSS_ENOENT	That account index does not exist
HPSS_EPERM	User is not a member of that account
HPSS_ESYSTEM	This routine received an internal eror; usually a site customization problem.
<0	Some other error.

See Also

av_default_ValidateAccount

Note

The OkToCache should normally be TRUE. If a site customizes this routine it should only set this argument to FALSE if the returned informatio can change over time and should not be cached by the client.

```
av_ValidateAccount() {
        if site style and creating a file and account inheritance is on {
                   if (parent account supplied and
                           (Acct is ACCT_REC_DEFAULT or parent account)) {
                   return with parent account;
           } else if (no parent account supplied and
                      Acct != ACCT_REC_DEFAULT) {
                //  assume caller (Core Server) has determined parent already
                return with Acct;
            }
         }
         if (user is superuser)
            return with Acct;
         if (user is "nobody") {
            if auth caller
               return with ACCT_REC_DEFAULT
          return HPSS_EPERM;
         }
         if site style accounting {
         Read the account from metadata.
         If (not found and Acct == ACCT_REC_DEFAULT and auth caller) {
                 Create default account for this user;
         } else if not found {
                 error = HPSS_EPERM;
         }
      } else if unix style {
         if Acct == UID or ACCT_REC_DEFAULT or auth caller
                 return with Uid;
         error = HPSS_EPERM;
      }
      return error;
}
```

9.37.2.6 signed32 av_site_ValidateChacct (hpss_connect_handle_t *Binding*, hpss_reqid_t *RequestId*, unsigned32 *UserRealmId*, unsigned32 *UserUid*, unsigned32 *FileRealmId*, unsigned32 *FileUid*, unsigned32 *FileGid*, acct_rec_t *OldAcct*, acct_rec_t *NewAcct*, acct_rec_t ∗ *OutAcct*, unsigned32 ∗ *OkToCache*)

Determine account change index.

This site customizable function is called by the client API to determine which account index to use when changing the account index of a file or directory (i.e. hpss_Chacct). Implementation is up to the individual site. If this routine is not implemented by a site, it should return HPSS_EUSEDEFAULT to tell account validation to perform the default behavior.

Parameters

in	*Binding*	RPC binding information
in	*RequestId*	Id of this Request
in	*UserRealmId*	Realm Id of the User Performing This Op
in	*UserUid*	User Id (unix uid) of the User Perfoming This Op
in	*FileRealmId*	Realm Id of the file owner
in	*FileUid*	User Id (unix uid) of the file owner
in	*FileGid*	Group Id (unix gid) of the file
in	*OldAcct*	Old account index for the file
in	*NewAcct*	New account index for the file
out	*OutAcct*	Account index for the file
out	*OkToCache*	TRUE if this information may be cached by the client.

Return values

HPSS_EUSEDEFAULT	If the routine is not implemented by the site, use the corresponding default routine
0	Success
HPSS_EBUSY	The server is busy. The reqest should be tried again later.
HPSS_ENOENT	That account index does not exist
HPSS_EPERM	User is not a member of that account
HPSS_ESYSTEM	This routine received an internal eror; usually a site customization problem.
<0	Some other error.

See Also

av_default_ValidateChacct

Note

The OkToCache should normally be TRUE. If a site customizes this routine it should only set this argument to FALSE if the returned informatio can change over time and should not be cached by the client.

```
av_ValidateChacct() {
        if user is "nobody", return with *OutAcct = ACCT_REC_DEFAULT;
        if site style accounting {
            if user is superuser and Acct != ACCT_REC_DEFAULT
                   return with NewAcct;
            error = read user account from metadata
            if not found {
                   if NewAcct == ACCT_REC_DEFAULT, return 0
                   else return HPSS_EPERM;
             }
         } else if unix style accounting {
             if NewAcct is FileUid or ACCT_REC_DEFAULT
                     return with *OutAcct = FileUid;
             error = HPSS_EPERM;
         }
         return error;
}
```

9.37.2.7 signed32 av_site_ValidateChown (hpss_connect_handle_t *Binding,* hpss_reqid_t *RequestId,* unsigned32 *OldRealmId,* unsigned32 *OldUid,* unsigned32 *OldGid,* acct_rec_t *OldAcct,* unsigned32 *NewRealmId,* unsigned32 *NewUid,* unsigned32 *NewGid,* acct_rec_t *SessionAcct,* acct_rec_t ∗ *OutAcct,* unsigned32 ∗ *OkToCache*)

Determine file ownership index.

This site customizable function is called by the client API to determine which account index to use when changing the owner of a file or directory (i.e. hpss_Chown, various set attributes calls, etc.). Implementation is up to the individual site. If this routine is not implemented by a site, it should return HPSS_EUSEDEFAULT to tell account validation to perform the default behavior.

Parameters

in	*Binding*	RPC binding information
in	*RequestId*	Id of this Request
in	*OldRealmId*	Old Realm Id of the file owner
in	*OldUid*	Old User Id (unix uid) of the file owner
in	*OldGid*	Old Group Id (unix gid) of the file
in	*OldAcct*	Old account index of the file
in	*NewRealmId*	New Realm Id of the file owner
in	*NewUid*	New User Id (unix uid) of the file owner
in	*NewGid*	New Group Id (unix gid) of the file
in	*SessionAcct*	Current session account index of user
out	*OutAcct*	Account index to use for the file
out	*OkToCache*	TRUE if this information may be cached by the client.

Return values

HPSS_EUSEDEFAULT	If the routine is not implemented by the site, use the corresponding default routine
0	Success
HPSS_EBUSY	The server is busy. The reqest should be tried again later.
HPSS_ENOENT	That account index does not exist
HPSS_EPERM	User is not a member of that account
HPSS_ESYSTEM	This routine received an internal eror; usually a site customization problem.
<0	Some other error.

See Also

av_default_ValidateChown

Note

The OkToCache should normally be TRUE. If a site customizes this routine it should only set this argument to FALSE if the returned informatio can change over time and should not be cached by the client.

```
av_ValidateChown() {
        if user is "nobody", return with *OutAcct = ACCT_REC_DEFAULT;
        if site style accounting {
                error = read user account from metadata
                if not found {
                        error = 0;
                         OutAcct = ACCT_REC_DEFAULT;
                }
           } else if unix style accounting {
                  OutAcct = NewUid;
            }
         return error;
    }
```

9.37.2.8 signed32 av_site_ValidateCreate (hpss_connect_handle_t *Binding*, hpss_reqid_t *RequestId*, unsigned32 *RealmId*, unsigned32 *Uid*, unsigned32 *Gid*, acct_rec_t *Acct*, acct_rec_t *ParentAcct*, hpss_Attrs_t ∗ *ParentAttrs*, acct_rec_t ∗ *OutAcct*, unsigned32 ∗ *OkToCache*)

Determine new file/directory account index.

This site customizable function is called by the client API to determine which account index to use when creating a new file or directory (i.e. hpss_Create). Implementation is up to the individual site. If this routine is not implemented by a site, it should return HPSS_EUSEDEFAULT to tell account validation to perform the default behavior.

Parameters

in	*Binding*	RPC binding information
in	*RequestId*	Id of this Request
in	*RealmId*	New file owner's Realm Id
in	*Uid*	User Id (unix uid) of the new file's owner
in	*Gid*	Group Id (unix gid) of the new file
in	*Acct*	Account index of the new file
in	*ParentAcct*	Account index of the new file's parent directory
in	*ParentAttrs*	Parent Directory Attributes including: • CORE_ATTR_MODE_PERMS
out	*OutAcct*	Account index to use for the file
out	*OkToCache*	TRUE if this information may be cached by the client.

Return values

HPSS_EUSEDEFAULT	If this routine is not implemented by the site, use the corresponding default routine.
0	Success
HPSS_EBUSY	The server is busy. The reqest should be tried again later.
HPSS_ENOENT	That account index does not exist
HPSS_EPERM	User is not a member of that account
HPSS_ESYSTEM	This routine received an internal eror; usually a site customization problem.
<0	Some other error.

See Also

av_default_ValidateCreate

Note

The OkToCache should normally be TRUE. If a site customizes this routine it should only set this argument to FALSE if the returned informatio can change over time and should not be cached by the client.

```
av_ValidateCreate() {
    if site style accounting and account inheritance and
       parent account is supplied {
       return with parent account;
    }
    if user is "nobody", return with ACCT_REC_DEFAULT;
    if site style accounting {
       if (user is superuser and Acct != ACCT_REC_DEFAULT)
               return ok with OutAcct = Acct;
        error = read user account from metadata;
        if not found {
                if Acct == ACCT_REC_DEFAULT, return with OutAcct = Acct;
                else return with HPSS_EPERM;
        }
     } else if unix style accounting {
              OutAcct = Uid;
     }
     log any error;
     return error;
}
```

Chapter 10

Class Documentation

10.1 hpss_Attrs Struct Reference

Core Server Object Attribute Structure.

Public Attributes

- acct_rec_t Account
- bf_activity_count_t Activity
- bfs_bitfile_obj_handle_t BitfileObj
- fstring Comment [256]
- uint32_t CompositePerms
- uint32_t COSId
- uint64_t DataLength
- uint32_t EntryCount

    Number of entries in a directory.
- uint32_t ExtendedACLs

    Whether the object has extended ACLs.
- uint32_t FamilyId
- ns_ObjHandle_t FilesetHandle
- uint64_t FilesetId
- uint64_t FilesetRootObjectId
- uint32_t FilesetStateFlags

    Contains flag bits indicating the state of the fileset. The following constants define the possible states:
- uint32_t FilesetType

    Specifies the fileset type.
- uint32_t GID
- uint32_t GroupPerms
- uint32_t LinkCount
- uint32_t ModePerms
- uint32_t OpenCount
- uint32_t OptionFlags
- uint32_t OtherPerms
- uint32_t ReadCount
- uint32_t RealmId
- uint64_t RegisterBitMap
- uint32_t SubSystemId

- timestamp_sec_t TimeCreated
- timestamp_sec_t TimeLastRead
- timestamp_sec_t TimeLastWritten
- timestamp_sec_t TimeModified
- hpss_TrashRecord_t TrashInfo
- uint32_t Type

  Specifies the object type.
- uint32_t UID
- uint32_t UserPerms
- uint32_t WriteCount

10.1.1 Detailed Description

Core Server Object Attribute Structure.

10.1.2 Member Data Documentation

10.1.2.1 acct_rec_t hpss_Attrs::Account

Opaque acounting info

10.1.2.2 bf_activity_count_t hpss_Attrs::Activity

Current bitfile activity (getxattrs only)

10.1.2.3 bfs_bitfile_obj_handle_t hpss_Attrs::BitfileObj

Bitfile Object

10.1.2.4 fstring hpss_Attrs::Comment[256]

Uninterpreted client-supplied ASCII text

10.1.2.5 uint32_t hpss_Attrs::CompositePerms

Permission on an object after all ACLs have been examined and applied.

10.1.2.6 uint32_t hpss_Attrs::COSId

Class of Servie ID

10.1.2.7 uint64_t hpss_Attrs::DataLength

Data length in bytes

10.1.2.8 uint32_t hpss_Attrs::EntryCount

Number of entries in a directory.

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

10.1.2.9 uint32_t hpss_Attrs::ExtendedACLs

Whether the object has extended ACLs.

A flag that indicates whether an object has extended ACLs (ACL entries other han user_obj, group_obj, and other-_obj). If 1, the object has extended ACLs.

10.1.2.10 uint32_t hpss_Attrs::FamilyId

File family identifier

10.1.2.11 ns_ObjHandle_t hpss_Attrs::FilesetHandle

Object's root fileset handle

10.1.2.12 uint64_t hpss_Attrs::FilesetId

Object's fileset id

10.1.2.13 uint64_t hpss_Attrs::FilesetRootObjectId

The root id of the filset

10.1.2.14 uint32_t hpss_Attrs::FilesetStateFlags

Contains flag bits indicating the state of the fileset. The following constants define the possible states:

- CORE_FS_STATE_READ Read is permitted.
- CORE_FS_STATE_WRITE Write is permitted.
- CORE_FS_STATE_DESTROYED The fileset has been destroyed; neither read nor write will be permitted.
- CORE_FS_STATE_READ_WRITE A combination of READ and WRITE.
- CORE_FS_STATE_COMBINED A combination of all bit settings above.

10.1.2.15 uint32_t hpss_Attrs::FilesetType

Specifies the fileset type.

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

- CORE_FS_TYPE_HPSS_ONLY The fileset is an HPSS-only fileset.
- CORE_FS_TYPE_ARCHIVED The fileset is a backup copy of some other fileset (unused).
- CORE_FS_TYPE_NATIVE The fileset is native to some other file system such as XFS (unused).
- CORE_FS_TYPE_MIRRORED The fileset is a mirrored copy of some other fileset (unused).

10.1.2.16 uint32_t hpss_Attrs::GID

Principal group identifier

10.1.2.17 uint32_t hpss_Attrs::GroupPerms

Permissions granted to group members

10.1.2.18 uint32_t hpss_Attrs::LinkCount

Number of hard links to the file object

10.1.2.19 uint32_t hpss_Attrs::ModePerms

Flags field which controls the SetUID, SetGID, and SetSticky

10.1.2.20 uint32_t hpss_Attrs::OpenCount

Number of opens to the file object

10.1.2.21 uint32_t hpss_Attrs::OptionFlags

Flag whih controls the DontPurge option

10.1.2.22 uint32_t hpss_Attrs::OtherPerms

Permissions granted to ’other’ clients

10.1.2.23 uint32_t hpss_Attrs::ReadCount

Number of times the bitfile was read

10.1.2.24 uint32_t hpss_Attrs::RealmId

Client Realm Id

10.1.2.25 uint64_t hpss_Attrs::RegisterBitMap

Which fields should trigger SSM notifications

10.1.2.26 uint32_t hpss_Attrs::SubSystemId

Subsystem Id

10.1.2.27 timestamp_sec_t hpss_Attrs::TimeCreated

Time the object was created

10.1.2.28 timestamp_sec_t hpss_Attrs::TimeLastRead

Time the object was last accessed

10.1.2.29 timestamp_sec_t hpss_Attrs::TimeLastWritten

Time the object was last written

10.1.2.30 timestamp_sec_t hpss_Attrs::TimeModified

Time the metadata was last modified

10.1.2.31 hpss_TrashRecord_t hpss_Attrs::TrashInfo

Trashcan structures; only valid if the file is trashcanned

10.1.2.32 uint32_t hpss_Attrs::Type

Specifies the object type.

This field is not settable. Specifies the type of the object:

- NS_OBJECT_TYPE_DIRECTORY Directory
- NS_OBJECT_TYPE_FILE Regular File
- NS_OBJECT_TYPE_JUNCTION Junction
- NS_OBJECT_TYPE_SYM_LINK Symbolic Link
- NS_OBJECT_TYPE_HARD_LINK Hard Link

10.1.2.33 uint32_t hpss_Attrs::UID

User identifier of the object's owner

10.1.2.34 uint32_t hpss_Attrs::UserPerms

Permissions granted to the owner of the object

10.1.2.35 uint32_t hpss_Attrs::WriteCount

Number of writes to a file object

The documentation for this struct was generated from the following file:

- hpss_attrs.x

Chapter 11

File Documentation

11.1 acct_Site.c File Reference

```
#include "hpss_limits.h"
#include "hpss_errno.h"
#include "acct_hpss.h"
#include "acct_site.h"
```

Functions

- signed32 av_site_AcctIdxToName (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,unsigned32 Gid,unsigned32 Flags,acct_rec_t Acct,char ∗AcctName,unsigned32 ∗OkToCache)

  Convert the supplied account index into an account name.
- signed32 av_site_AcctNameToIdx (hpss_connect_handle_t Binding, hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,unsigned32 Gid,unsigned32 Flags,char ∗InAcctName,char ∗OutAcctName,acct_-rec_t ∗Acct,unsigned32 ∗OkToCache)

  Convert account name to account index.
- signed32 av_site_Initialize (acct_config_t ∗AcctPolicy)

  Initialize site code.
- signed32 av_site_Shutdown (void)

  Shutdown site code.
- signed32 av_site_ValidateAccount (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,acct_rec_t Acct,unsigned32 Flags,acct_rec_t ∗ParentAcct,acct_rec_t ∗Out-Acct,unsigned32 ∗OkToCache)

  Validate a user's account index.
- signed32 av_site_ValidateChacct (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 UserRealmId,unsigned32 UserUid,unsigned32 FileRealmId,unsigned32 FileUid,unsigned32 FileGid,acct_-rec_t OldAcct,acct_rec_t NewAcct,acct_rec_t ∗OutAcct,unsigned32 ∗OkToCache)

  Determine account change index.
- signed32 av_site_ValidateChown (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 Old-RealmId,unsigned32 OldUid,unsigned32 OldGid,acct_rec_t OldAcct,unsigned32 NewRealmId,unsigned32 NewUid,unsigned32 NewGid,acct_rec_t SessionAcct,acct_rec_t ∗OutAcct,unsigned32 ∗OkToCache)

  Determine file ownership index.
- signed32 av_site_ValidateCreate (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,unsigned32 Gid,acct_rec_t Acct,acct_rec_t ParentAcct,hpss_Attrs_t ∗Parent-Attrs,acct_rec_t ∗OutAcct,unsigned32 ∗OkToCache)

  Determine new file/directory account index.

11.2 acct_site.h File Reference

```
#include "hpss_rpc.h"
#include "hpss_attrs.h"
#include "acct_hpss.h"
#include "acct_config.h"
```

Functions

- signed32 av_site_AcctIdxToName (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,unsigned32 Gid,unsigned32 Flags,acct_rec_t Acct,char ∗AcctName,unsigned32 ∗OkToCache)

  Convert the supplied account index into an account name.
- signed32 av_site_AcctNameToIdx (hpss_connect_handle_t Binding, hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,unsigned32 Gid,unsigned32 Flags,char ∗InAcctName,char ∗OutAcctName,acct_-rec_t ∗Acct,unsigned32 ∗OkToCache)

  Convert account name to account index.
- signed32 av_site_Initialize (acct_config_t ∗AcctPolicy)

  Initialize site code.
- signed32 av_site_Shutdown (void)

  Shutdown site code.
- signed32 av_site_ValidateAccount (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,acct_rec_t Acct,unsigned32 Flags,acct_rec_t ∗ParentAcct,acct_rec_t ∗Out-Acct,unsigned32 ∗OkToCache)

  Validate a user's account index.
- signed32 av_site_ValidateChacct (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 UserRealmId,unsigned32 UserUid,unsigned32 FileRealmId,unsigned32 FileUid,unsigned32 FileGid,acct_-rec_t OldAcct,acct_rec_t NewAcct,acct_rec_t ∗OutAcct,unsigned32 ∗OkToCache)

  Determine account change index.
- signed32 av_site_ValidateChown (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 Old-RealmId,unsigned32 OldUid,unsigned32 OldGid,acct_rec_t OldAcct,unsigned32 NewRealmId,unsigned32 NewUid,unsigned32 NewGid,acct_rec_t SessionAcct,acct_rec_t ∗OutAcct,unsigned32 ∗OkToCache)

  Determine file ownership index.
- signed32 av_site_ValidateCreate (hpss_connect_handle_t Binding,hpss_reqid_t RequestId,unsigned32 RealmId,unsigned32 Uid,unsigned32 Gid,acct_rec_t Acct,acct_rec_t ParentAcct,hpss_Attrs_t ∗Parent-Attrs,acct_rec_t ∗OutAcct,unsigned32 ∗OkToCache)

  Determine new file/directory account index.

11.3 api_access.c File Reference

Functions for determining file access.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Access (const char ∗Path, int Amode)

  Checks the accessibility of a file.
- int hpss_AccessHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, int Amode, const sec_cred_t ∗Ucred)

  Checks the accessibility of a file using a handle.

11.3.1 Detailed Description

Functions for determining file access.

11.4 api_acct.c File Reference

Functions for manipulating and translating account information.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
#include "acct_av_lib.h"
#include "hpss_uuid.h"
```

Functions

- int API_DetermineAcct (const sec_cred_t ∗UserCred, apithrdstate_t ∗ThreadContext, hpss_srvr_id_t Core-ServerID, hpss_reqid_t RequestID, hpss_id_t ∗SiteId, acct_rec_t ∗AcctCode)

  Determine a session account.
- int API_GetAcctForParent (apithrdstate_t ∗ThreadContext, hpss_reqid_t RequestID, const ns_ObjHandle_t ∗ParentHandle, acct_rec_t ∗AcctCode)

  Retrieves a parent handle's cached account code.
- int API_PutAcctForParent (ns_ObjHandle_t ∗ParentHandle, acct_rec_t AcctCode)

  Caches an account code for a parent handle.
- int API_SetCurrentSessionAcct (apithrdstate_t ∗ThreadContext, const hpss_id_t ∗SiteId)

  Changes the accounting state of a thread.
- int hpss_AcctCodeToName (acct_rec_t AcctCode, hpss_id_t ∗Site, char ∗AcctName)

  Maps an account code to its corresponding account name.
- int hpss_AcctNameToCode (char ∗AcctName, hpss_id_t ∗Site, acct_rec_t ∗AcctCode)

  Maps an account name to its corresponding account code.
- int hpss_Chacct (const char ∗Path, acct_rec_t AcctCode)

  Changes a file or directory's account code.
- int hpss_ChacctByName (const char ∗Path, const char ∗AcctName)

  Changes a file or directory's account name.
- int hpss_GetAcct (acct_rec_t ∗RetDefAcct, acct_rec_t ∗RetCurAcct)

  Retrieves current and default account codes.
- int hpss_GetAcctName (char ∗AcctName)

  Retrieves account name.
- int hpss_SetAcct (acct_rec_t NewCurAcct)

Sets the current account code.

- int hpss_SetAcctByName (const char ∗NewAcctName)

  Sets the current account name.

11.4.1 Detailed Description

Functions for manipulating and translating account information.

11.4.2 Function Documentation

11.4.2.1 int API_DetermineAcct (const sec_cred_t ∗ *UserCred,* apithrdstate_t ∗ *ThreadContext,* hpss_srvr_id_t *CoreServerID,* hpss_reqid_t *RequestID,* hpss_id_t ∗ *SiteId,* acct_rec_t ∗ *AcctCode*)

Determine a session account.

The 'API_DetermineAcct' function returns the appropriate session account that should be used at the site managing the object referenced by the ns_parent_obj_handle. This function adds the new accounting information to the thread's account list, but not at the list beginning where the thread's current session account information is stored.

Parameters

in	*UserCred*	user credentials
in	*ThreadContext*	thread context
in	*CoreServerID*	id of Core Server
in	*RequestID*	request id
in	*SiteId*	sites location
out	*AcctCode*	account code

Return values

0	No error.
-ENOMEM	Unable to allocate memory.
-EFAULT	Invalid pointer encountered.

Note

> The first entry in the thread's account list is the current session account. The thread should have at least one entry already existing in its account list.

If the UserCreds passed in differ from the credentials stored in the thread context, the account from the user credentials is assumed to be correct and is passed back. This only happens for special authorized callers, like NFS/DFS, who don't really need to track current session accounts anyway. If we need to track sessions for separate users this in the future, we'll probably need to keep a separate account list for each cell-id/uid pair.

Note

> Immediately after this function returns, an Account Validation routine should be called to map the session account code to an actual account code in use.

Usually, this routine will be called after API_TraversePath since traversing the path to the object could easily have taken us out of the site that manages the current working directory (which means that the current session account information in the threadstate will not be applicable to the object).

11.4.2.2 int API_GetAcctForParent (apithrdstate_t ∗ *ThreadContext,* hpss_reqid_t *RequestID,* const ns_ObjHandle_t ∗ *ParentHandle,* acct_rec_t ∗ *AcctCode*)

Retrieves a parent handle's cached account code.

The 'API_GetAcctForParent' function is called to get a cached account code for the specified parent. If there isn't a cached account code for the specified parent, then it is retrieved from the Core Server, cached and returned.

Parameters

`in`	*ThreadContext*	thread context
`in`	*RequestID*	request id
`in`	*ParentHandle*	directory handle
`out`	*AcctCode*	account code

Return values

0	- No error. Account code is returned
-ENOMEM	Unable to allocate memory.
-EINVAL	Not a directory handle
Other	Return codes from API_TraversePath()

11.4.2.3 int API_PutAcctForParent (ns_ObjHandle_t ∗ *ParentHandle,* acct_rec_t *AcctCode*)

Caches an account code for a parent handle.

The 'API_PutAcctForParent' function is called to cache the specifed account code for the specified parent handle.

Parameters

`in`	*ParentHandle*	directory handle
`in`	*AcctCode*	account code

Return values

-EINVAL	Not a directory handle
-ENOMEM	Unable to allocate memory.
0	No error. Account code is cached

11.4.2.4 int API_SetCurrentSessionAcct (apithrdstate_t ∗ *ThreadContext,* const hpss_id_t ∗ *SiteId*)

Changes the accounting state of a thread.

The 'API_SetCurrentSessionAcct' function changes the accounting state of a thread to a state that is appropriate for the SiteId.

Parameters

`in,out`	*ThreadContext*	thread context
`in`	*SiteId*	site id

Return values

-EFAULT	SiteId or threadcontext pointer is not valid.
0	No error. State of the thread is changed.

Note

This function exists to change the accounting state of the current thread to a state appropriate for SiteId. It should be called after or during any operation that changes the current site.

11.5 api_acl.c File Reference

Functions for manipulating access control lists.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_DeleteACL (const char ∗Path, const uint32_t Options, const ns_ACLConfArray_t ∗ACL)

  Removes an ACL entry from a file or directory.
- int hpss_DeleteACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, const uint32_t Options, const ns_ACLConfArray_t ∗ACL)

  Removes an ACL entry from a file or directory using a handle.
- int hpss_GetACL (const char ∗Path, uint32_t Options, ns_ACLConfArray_t ∗ACL)

  Retrives the ACL of a file or directory.
- int hpss_GetACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, const uint32_t Options, ns_ACLConfArray_t ∗ACL)

  Retrieves an ACL of a file or directory using a handle.
- int hpss_SetACL (const char ∗Path, const uint32_t Options, const ns_ACLConfArray_t ∗ACL)

  Sets the ACL entries of a file or directory.
- int hpss_SetACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, const uint32_t Options, const ns_ACLConfArray_t ∗ACL)

  Sets the ACL entries of a file or directory using a handle.
- int hpss_UpdateACL (const char ∗Path, int32_t Options, const ns_ACLConfArray_t ∗ACL)

  Updates an ACL array of a file or directory.
- int hpss_UpdateACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t Options, const ns_ACLConfArray_t ∗ACL)

  Updates an ACL entry of a file or directory using a handle.

11.5.1 Detailed Description

Functions for manipulating access control lists.

11.6 api_cache.c File Reference

Functions for caching information.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "u_signed64.h"
#include <stdbool.h>
#include "hpss_api.h"
#include "api_internal.h"
#include "hpss_irbtree.h"
#include <inttypes.h>
#include <pthread.h>
```

Functions

- int32_t API_AddFSDirCache (const char ∗User, const u_signed64 FSID, const ns_ObjHandle_t ∗TrashHandle, const ns_ObjHandle_t ∗TrashContainer)

  Add an entry into the fs cache.
- int32_t API_AddHomeDirCache (const char ∗User, const u_signed64 FSID, const ns_ObjHandle_t ∗TrashHandle, const ns_ObjHandle_t ∗TrashContainer)

  Add an entry into the homedir cache.
- int32_t API_ClearFSDirCache ()

  Remove all entries from the cache.
- int32_t API_ClearHomeDirCache ()

  Remove all entries from the cache.
- int32_t API_GetFSDirCache (const char ∗User, const u_signed64 ∗FSID, ns_ObjHandle_t ∗TrashHandle, ns_ObjHandle_t ∗TrashContainer)

  Retrieve a fs cache entry.
- int32_t API_GetHomeDirCache (const char ∗User, u_signed64 ∗FSID, ns_ObjHandle_t ∗TrashHandle, ns_ObjHandle_t ∗TrashContainer)

  Retrieve a homedir cache entry.
- int32_t API_GetJunctionCache (const ns_ObjHandle_t ∗FilesetHandle, hpss_junction_ent_t ∗Junc, hpss_fileattr_t ∗Attrs)

  Get junction cache entry.
- int32_t API_InitFSDirCache ()

  Initialize the fs trashcan cache.
- int32_t API_InitHomeDirCache ()

  Initialize the home directory trashcan cache.
- int32_t API_RebuildJunctionCache ()

  Recreate the junction cache.
- int32_t API_RemoveFSDirCache (const char ∗User, const u_signed64 ∗FSID)

  Remove an entry from the fs cache.
- int32_t API_RemoveHomeDirCache (const char ∗User)

  Remove an entry from the homedir cache.
- int fsrootkeycompare (const hpss_irbtree_node_t ∗t1, const hpss_irbtree_node_t ∗t2)

  Compare two cached fs handles.
- int junction_cache_entry_compare (const hpss_irbtree_node_t ∗t1, const hpss_irbtree_node_t ∗t2)

  Compare two cached junction entries.
- int32_t stale_thresh_cmp (const hpss_irbtree_node_t ∗d1, const hpss_irbtree_node_t ∗d2)

Calls stale_thresh_cmp_factor() with a factor of 1.

- int32_t stale_thresh_cmp_factor (trash_cache_t ∗t1, trash_cache_t ∗t2, int32_t factor)

    Uses the cache stale cutoff time to determine staleness.
- int32_t stale_thresh_cmp_half (const hpss_irbtree_node_t ∗d1, const hpss_irbtree_node_t ∗d2)

    Calls stale_thresh_cmp_factor() with a factor of 2.
- int32_t stale_thresh_cmp_quarter (const hpss_irbtree_node_t ∗d1, const hpss_irbtree_node_t ∗d2)

    Calls stale_thresh_cmp_factor() with a factor of 4.
- int32_t thresh_cmp (trash_cache_t ∗entry, time_t cutoff)

    Determines whether an trash cache entry is stale.
- int32_t userkeycompare (const hpss_irbtree_node_t ∗d1, const hpss_irbtree_node_t ∗d2)

    Compare two cached homedir handles.

11.6.1 Detailed Description

Functions for caching information. This file contains methods for caching trashcan info. It utilizes a tree structure in order to maintain the cache, which allows for fast insert/delete/search times. The cache is limited by size, and will try to stale out old entries if the cache grows too large.

Also contains junction caching structures; the junction cache initializes upon the first request and stays in memory from then on. If a cache miss occurs the cache is reinitialized.

11.6.2 Function Documentation

11.6.2.1 int32_t API_AddFSDirCache (const char ∗ *User,* const u_signed64 *FSID,* const ns_ObjHandle_t ∗ *TrashHandle,* const ns_ObjHandle_t ∗ *TrashContainer*)

Add an entry into the fs cache.

Attempts to add an entry into the fs cache; if the cache is FULL then it first attempts to purge the cache down. If the cache remains full the entry is not added.

Note that the FSID, TrashHandle, and TrashContainer should all be valid. The caching expects a complete accounting of the trashcan info.

Parameters

in	*User*	Ucred username
in	*FSID*	Deleted Object's FSID
in	*TrashHandle*	Trashcan Directory Handle
in	*TrashContainer*	Trashcan Container (note for fs this is the fsroot)

Return values

HPSS_E_NOERROR	Success
HPSS_EINVAL	Cache is not initialized or invalid fsid
HPSS_ENOMEM	Out of memory
HPSS_ENOSPACE	Cache is full; couldn't purge down

11.6.2.2 int32_t API_AddHomeDirCache (const char ∗ *User,* const u_signed64 *FSID,* const ns_ObjHandle_t ∗ *TrashHandle,* const ns_ObjHandle_t ∗ *TrashContainer*)

Add an entry into the homedir cache.

Attempts to add an entry into the homedir cache; if the cache is FULL then it first attempts to purge the cache down. If the cache remains full the entry is not added.

Note that the FSID, TrashHandle, and TrashContainer should all be valid. The caching expects a complete accounting of the trashcan info.

Parameters

in	*User*	Ucred username
in	*FSID*	Homedir Fileset Id
in	*TrashHandle*	Trashcan Directory Handle
in	*TrashContainer*	Trashcan Container (note for homedirs this is the homedir)

Return values

HPSS_E_NOERROR	Success
HPSS_EINVAL	Cache is not initialized or invalid fsid
HPSS_ENOMEM	Out of memory
HPSS_ENOSPACE	Cache is full; couldn't purge down

11.6.2.3 int32_t API_ClearFSDirCache ( )

Remove all entries from the cache.

Return values

HPSS_E_NOERROR	Success
HPSS_EINAL	Cache is not initialized

Note

This function completely frees all cache resources, and then reinitializes an empty tree for future processing.

11.6.2.4 int32_t API_ClearHomeDirCache ( )

Remove all entries from the cache.

Return values

HPSS_E_NOERROR	Success
HPSS_EINVAL	Cache is not initialized

Note

This function completely frees all cache resources, and then reinitializes an empty tree for future processing.

11.6.2.5 int32_t API_GetFSDirCache (const char ∗ *User,* const u_signed64 ∗ *FSID,* ns_ObjHandle_t ∗ *TrashHandle,* ns_ObjHandle_t ∗ *TrashContainer*)

Retrieve a fs cache entry.

Retrieves a fs cache entry; the last used time is updated to keep the entry from being staled out of the cache. The entry remains in the cache.

Parameters

`in`	*User*	User ucred; This a partial lookup key
`in`	*FSID*	Parent FSID; partial lookup key
`in,out`	*TrashHandle*	Trashcan Handle
`in,out`	*TrashContainer*	Trashcan Container

Return values

HPSS_EINVAL	Cache is not initialized
HPSS_ENOENT	No entry found
HPSS_E_NOERROR	Success

11.6.2.6 int32_t API_GetHomeDirCache (const char ∗ *User,* u_signed64 ∗ *FSID,* ns_ObjHandle_t ∗ *TrashHandle,* ns_ObjHandle_t ∗ *TrashContainer*)

Retrieve a homedir cache entry.

Retrieves a homedir cache entry; the last used time is updated to keep the entry from being staled out of the cache. The entry remains in the cache.

Parameters

`in`	*User*	User ucred; This is the lookup key
`in,out`	*FSID*	Fileset Id
`in,out`	*TrashHandle*	Trashcan Handle
`in,out`	*TrashContainer*	Trashcan Container

Return values

HPSS_EINVAL	Cache is not initialized
HPSS_ENOENT	No entry found
HPSS_E_NOERROR	Success

11.6.2.7 int32_t API_GetJunctionCache (const ns_ObjHandle_t ∗ *FilesetHandle,* hpss_junction_ent_t ∗ *Junc,* hpss_fileattr_t ∗ *Attrs*)

Get junction cache entry.

Note

This function will automatically build the cache if it has not been done yet.

11.6.2.8 int32_t API_InitFSDirCache ( )

Initialize the fs trashcan cache.

Return values

HPSS_E_NOERROR	Success
HPSS_ENOMEM	Out of memory

11.6.2.9 int32_t API_InitHomeDirCache ( )

Initialize the home directory trashcan cache.

Return values

HPSS_E_NOERROR	Success
HPSS_ENOMEM	Out of memory

11.6.2.10 int32_t API_RebuildJunctionCache ( )

Recreate the junction cache.

This function releases the junction cache and then rebuilds it in full.

11.6.2.11 int32_t API_RemoveFSDirCache (const char * *User,* const u_signed64 * *FSID*)

Remove an entry from the fs cache.

Attempts to remove an entry from the fs cache.

Note that the FSID, must be valid.

Parameters

`in`	*User*	Ucred username
`in`	*FSID*	Deleted object's fileset id

Return values

HPSS_E_NOERROR	Success
HPSS_EINVAL	Cache is not initialized
HPSS_ENOMEM	Out of memory

11.6.2.12 int32_t API_RemoveHomeDirCache (const char * *User*)

Remove an entry from the homedir cache.

Attempts to remove an entry from the homedir cache.

Note that the User should be valid.

Parameters

`in`	*User*	Ucred username

Return values

HPSS_E_NOERROR	Success
HPSS_EINVAL	Cache is not initialized
HPSS_ENOMEM	Out of memory

11.6.2.13 int fsrootkeycompare (const hpss_irbtree_node_t * *t1,* const hpss_irbtree_node_t * *t2*)

Compare two cached fs handles.

Parameters

in	*t1*	irbTree node that intrudes upon first fs cache
in	*t2*	irbTree node that intrudes upon second fs cache

Return values

0	cache entries are equal
-1	cache entry 1 comes before 2
1	cache entry 1 comes after 2

Note

The fs cache has a dual key - the user name and the fsid. This is because in a fs trash scenario each user gets their own trashcan in the fs root trash.

11.6.2.14 int32_t stale_thresh_cmp_factor (trash_cache_t ∗ *t1,* trash_cache_t ∗ *t2,* int32_t *factor*)

Uses the cache stale cutoff time to determine staleness.

This is the default stale comparison function. It uses the provided stale time to clean out old entries to make room for new entries.

Parameters

in	*t1*	irbTree node that corresponds to trash cache entry
in	*t2*	irbTree node object provided by the arbitrary comparison function (unused)
in	*factor*	cutoff factor, increase to lower the cutoff time

Return values

FALSE	stale
TRUE	not stale

11.6.2.15 int32_t thresh_cmp (trash_cache_t ∗ *entry,* time_t *cutoff*)

Determines whether an trash cache entry is stale.

Parameters

in	*entry*	trash cache entry
in	*cutoff*	Time past an entry would be stale

Return values

FALSE	stale
TRUE	not stale

11.6.2.16 int32_t userkeycompare (const hpss_irbtree_node_t ∗ *d1,* const hpss_irbtree_node_t ∗ *d2*)

Compare two cached homedir handles.

Parameters

in	*d1*	irbTree node that intrudes upon first cache entry
in	*d2*	irbTree node that intrudes upon second cache entry

Return values

0	names are equal
-1	h1 comes before h2
1	h1 comes after h2

11.7 api_chdir.c File Reference

Functions for changing directories.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Chdir (const char ∗Path)

  Changes the current working directory.

11.7.1 Detailed Description

Functions for changing directories.

11.8 api_chmod.c File Reference

Functions for modifying object mode bits.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Chmod (const char ∗Path, mode_t Mode)

  Alters a file or directory's permissions.

11.8.1 Detailed Description

Functions for modifying object mode bits.

11.9 api_chown.c File Reference

Functions for changing the owner and/or group of an object.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
#include "acct_av_lib.h"
```

Functions

- int hpss_Chown (const char ∗Path, uid_t Owner, gid_t Group)

  Changes the user id and group id of a file or directory.

11.9.1 Detailed Description

Functions for changing the owner and/or group of an object.

11.10 api_chroot.c File Reference

Functions for modifying the default root location for a client.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int API_ApplyRootDir (const char ∗Path, const char ∗RootDir, char ∗NewPath, apithrdstate_t ∗ThreadContext)

  Apply the client's root directory.
- int hpss_Chroot (const char ∗Path)

  Sets the client's root directory.

11.10.1 Detailed Description

Functions for modifying the default root location for a client.

11.10.2 Function Documentation

11.10.2.1 int API_ApplyRootDir (const char ∗ *Path,* const char ∗ *RootDir,* char ∗ *NewPath,* apithrdstate_t ∗ *ThreadContext*)

Apply the client's root directory.

The 'ApplyRootDir' function applies the client's root directory to the passed pathname.

Parameters

`in`	*Path*	path to the object
`in`	*RootDir*	root dir path
`out`	*NewPath*	path rel root dir
`in`	*ThreadContext*	current context

Return values

-ENAMETOOLONG	The provided pathname is too long, or the constructed pathname is too long.

11.11 api_close.c File Reference

Functions for closing open files.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Close (int Fildes)

  Terminates the connection between a file and its handle.

11.11.1 Detailed Description

Functions for closing open files.

11.12 api_closedir.c File Reference

Functions for closing open directories.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Closedir (int Dirdes)

  Close an open directory stream.

11.12.1 Detailed Description

Functions for closing open directories.

11.13 api_copyfile.c File Reference

Functions for copying one file to another.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <pthread.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_CopyFile (int SrcFildes, int DestFildes)

  Copies the source file to the destination file.

11.13.1 Detailed Description

Functions for copying one file to another.

11.14 api_cos.c File Reference

Functions for setting the COS of a file.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_SetCOSByHints (int Fildes, uint32_t Flags, const hpss_cos_hints_t ∗HintsPtr, const hpss_cos_-priorities_t ∗PrioPtr, hpss_cos_md_t ∗COSPtr)

  Sets the COS of an empty file.

11.14.1 Detailed Description

Functions for setting the COS of a file.

11.15 api_distfile.c File Reference

Functions for retrieving or inserting file table information.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_GetDistFile (int Fildes, api_dist_file_info_t ∗FileInfo)

  Extracts a file table entry for an open file descriptor.
- int hpss_InsertDistFile (const api_dist_file_info_t ∗FileInfo)

  Inserts a file table entry into the current file table.

11.15.1 Detailed Description

Functions for retrieving or inserting file table information.

11.16 api_fgetattr.c File Reference

Functions for retrieving file attributes.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include "hpss_api.h"
#include "hpss_String_clnt.h"
#include "hpss_xml.h"
#include "api_internal.h"
#include "ss_pvlist.h"
#include "hpss_StringUtil.h"
```

Functions

- int API_ChecksumStateToFlags (const char ∗Value)

  Convert checksum state string to flags.
- int API_GetMultiFileDigest (int Fildes, uint64_t ∗StripeLength, hpss_file_hash_digest_list_t ∗DigestList)

  Retrieves a file's multi stripe hash digest, if any, for an open file.
- int API_GetMultiFileDigestHandle (apithrdstate_t ∗ThreadContext, const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, sec_cred_t ∗Ucred, uint64_t ∗StripeLength, hpss_file_hash_digest_list_t ∗DigestList)

  Retrieves a file's multi stripe hash digest from the handle, if any, for an open file.
- int API_GetSingleFileDigestHandle (apithrdstate_t ∗ThreadContext,const ns_ObjHandle_t ∗ObjHandle,const char ∗Path,sec_cred_t ∗Ucred,hpss_file_hash_digest_list_t ∗DigestList)

  Retrieves the file's hash digest, if any.
- const char ∗ API_UDAFindKey (const hpss_userattr_list_t ∗Attrs, char ∗Key)

  Retrieves the file's hash digest, if any, for an open file.
- int hpss_FgetFileDigest (int Fildes, hpss_file_hash_digest_t ∗Digest)

  Retrieves the file's hash digest, if any, for an open file.

- int hpss_FgetFileDigestList (int Fildes, hpss_file_hash_stripe_flags_t Flags, uint64_t ∗StripeLength, hpss_-file_hash_digest_list_t ∗DigestList)

  Retrieves a file's hash digest, if any, for an open file.
- int hpss_FileGetAttributes (const char ∗Path, hpss_fileattr_t ∗AttrOut)

  Queries the attributes of a file.
- int hpss_FileGetAttributesBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, char ∗Path, hpss_fileattr_t ∗Attr-Out)

  Get file attributes using a Bitfile Object.
- int hpss_FileGetAttributesHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, hpss_fileattr_t ∗AttrOut)

  Queries the attributes of a file using a handle.
- int hpss_FileGetDigestHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, sec_cred_t ∗Ucred, hpss_file_hash_digest_t ∗Digest)

  Retrieves the file's hash digest, if any.
- int hpss_FileGetDigestListHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, sec_cred_t ∗Ucred, hpss_file_hash_stripe_flags_t Flags, uint64_t ∗StripeLength, hpss_file_hash_digest_list_t ∗DigestList)

  Retrieves a file's hash digest(s) by preference, if any.
- int hpss_FileGetXAttributes (const char ∗Path, uint32_t Flags, uint32_t StorageLevel, hpss_xfileattr_t ∗Attr-Out)

  Queries the attributes of a file using flags and storage level perms.
- int hpss_FileGetXAttributesHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t Flags, uint32_t StorageLevel, hpss_xfileattr_t ∗AttrOut)

  Queries the attributes of a file using a handle, flags, and a storage level.
- int hpss_GetAttrHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗HandleOut, hpss_vattr_t ∗AttrOut)

  Obtains information about a file or directory using a handle.
- int hpss_GetFileNotrunc (const char ∗Path, int ∗NotruncFlag)

  Retrieves the file's NOTRUNC_FINAL_SEG flag.
- int hpss_GetRawAttrHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗HandleOut, hpss_vattr_t ∗AttrOut)

  Obtains information about a symlink or junction using a handle.

11.16.1 Detailed Description

Functions for retrieving file attributes.

11.16.2 Function Documentation

11.16.2.1 int API_ChecksumStateToFlags (const char ∗ *Value*)

Convert checksum state string to flags.

Returns the value for the provided key in the attribute list, if it exists. If the key exists multiple times, the first value in the attributes list is returned.

If the state is valid, the active and valid flags are set. If the state is invalid, the valid flag is still set to indicate that valid data was returned.

Parameters

`in`	*Value*	Checksum state string

Return values

>=0	Digest flags structure based upon the input state string
<0	Error indicating the nature of the error

11.16.2.2 int API_GetMultiFileDigest (int *Fildes,* uint64_t ∗ *StripeLength,* **hpss_file_hash_digest_list_t** ∗ *DigestList*)

Retrieves a file's multi stripe hash digest, if any, for an open file.

Returns the stripe file digest information. The caller allocates a buffer to hold the returned hash digest information and passes the pointer in the 'DigestList' parameter. On successful return, the supplied buffer is populated with the digest information.

Parameters

`in`	*Fildes*	Open bitfile descriptor
`out`	*StripeLength*	Digest stripe length
`out`	*DigestList*	Returned hash digest information

Return values

0	Success
-EBADF	Fildes is not a valid open descriptor
-ENOENT	Digest information doesn't exist
-EBUSY	Fildes is a busy descriptor
-ESTALE	Fildes is a stale descriptor and should be reopened
-EFAULT	DigestList output parameter is invalid

Note

The output buffer will contain the binary value for the hash digest.
The checksum buffers are stored in hex format and must be convered back to binary buffers.
The 'Flags' field of the digest information is used to indicate if a hash value is returned. The presence of the HPSS_FILE_HASH_DIGEST_VALID flag indicates that the digest buffer contains a valid hash value.

See Also

hpss_GetFileDigestHandle, hpss_FgetFileDigest

11.16.2.3 int API_GetMultiFileDigestHandle (apithrdstate_t ∗ *ThreadContext,* const **ns_ObjHandle_t** ∗ *ObjHandle,* const char ∗ *Path,* sec_cred_t ∗ *Ucred,* uint64_t ∗ *StripeLength,* **hpss_file_hash_digest_list_t** ∗ *DigestList*)

Retrieves a file's multi stripe hash digest from the handle, if any, for an open file.

Returns the stripe file digest information for a handle. The caller allocates a buffer to hold the returned hash digest information and passes the pointer in the 'DigestList' parameter. On successful return, the supplied buffer is populated with the digest information.

If invalid digest information exists, or no digest information exists, a striped digest populated with the 'hpss_hash_-type_none' type is returned to indicate that no valid checksum exists.

Since the UDA retrieve function returns ENOENT both for the non-existence of the attributes and the non-existence of the file, any existence checks should occur before this function is called.

Parameters

`in`	*ThreadContext*	API Thread Context
`in`	*ObjHandle*	Object Handle
`in`	*Path*	File Path
`in`	*Ucred*	User Credentials
`out`	*StripeLength*	Digest stripe length
`out`	*DigestList*	Returned hash digest information

Return values

0	Success
-EFAULT	DigestList output parameter is invalid

Note

The output buffer will contain the binary value for the hash digest.
The checksum buffers are stored in hex format and must be convered back to binary buffers.
The 'Flags' field of the digest information is used to indicate if a hash value is returned. The presence of the HPSS_FILE_HASH_DIGEST_VALID flag indicates that the digest buffer contains a valid hash value.

See Also

hpss_GetFileDigestHandle, hpss_FgetFileDigest

11.16.2.4 int API_GetSingleFileDigestHandle (apithrdstate_t ∗ *ThreadContext,* const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* sec_cred_t ∗ *Ucred,* hpss_file_hash_digest_list_t ∗ *DigestList*)

Retrieves the file's hash digest, if any.

Returns file digest information. Only single stripe file digest is retrieved. The caller allocates a buffer to hold the returned hash digest information and passes the pointer in the 'Digest' parameter. On successful return the supplied buffer is populated with the digest information.

Parameters

`in`	*ThreadContext*	API thread context
`in`	∗*ObjHandle*	parent object handle
`in`	∗*Path*	path of file to get attributes
`in`	∗*Ucred*	user credentials
`out`	∗*DigestList*	Returned hash digest information

Return values

0	Success
-ENOENT	File doesn't exist
-EINVAL	One or more of the parameters are invalid
-EFAULT	Digest output parameter is invalid

Note

The output buffer will contain the binary value for the hash digest.
A hash type of 'hpss_hash_type_none' will indicated the absence of a hash record.
The 'Flags' field of the digest information is used to indicate if a hash value is returned. The presence of the HPSS_FILE_HASH_DIGEST_VALID flag indicates that the digest buffer contains a valid hash value.

See Also

hpss_FgetFileDigest, hpss_FileGetDigestHandle

11.16.2.5 const char∗ API_UDAFindKey (const hpss_userattr_list_t ∗ *Attrs,* char ∗ *Key*)

Retrieves the file's hash digest, if any, for an open file.

Returns the value for the provided key in the attribute list, if it exists. If the key exists multiple times, the first value in the attributes list is returned.

Parameters

in	*Attrs*	User-defined Attributes
in	*Key*	Key to locate

Note

The key search is case sensitive.

Returns

The matching value in the attribute list, or NULL if the key does not exist.

11.17 api_fileset.c File Reference

Functions for manipulating filesets.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_FilesetCreate (const hpss_srvr_id_t ∗CoreServerID, uint32_t CreateOptions, ns_FilesetAttrBits-_t FilesetAttrBits, const ns_FilesetAttrs_t ∗FilesetAttrs, hpss_AttrBits_t ObjectAttrBits, const hpss_Attrs_-t ∗ObjectAttrs, ns_FilesetAttrBits_t RetFilesetAttrBits, hpss_AttrBits_t RetObjectAttrBits, ns_FilesetAttrs_t ∗RetFilesetAttrs, hpss_Attrs_t ∗RetObjectAttrs, ns_ObjHandle_t ∗FilesetHandle)

  Creates a new fileset.
- int hpss_FilesetDelete (const char ∗FilesetName, const uint64_t ∗FilesetId, const ns_ObjHandle_t ∗Fileset-Handle)

  Deletes an existing fileset.
- int hpss_FilesetGetAttributes (const char ∗FilesetName, const uint64_t ∗FilesetId, const ns_ObjHandle_-t ∗FilesetHandle, const hpss_srvr_id_t ∗CoreServerID, ns_FilesetAttrBits_t FilesetAttrBits, ns_FilesetAttrs_t ∗FilesetAttrs)

  Retrieves the attributes of a fileset.
- int hpss_FilesetListAll (uint64_t OffsetIn, uint32_t Entries, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_global_-fsent_t ∗FSentPtr)

  Retrieves attributes for every HPSS fileset.

- int hpss_FilesetSetAttributes (const char ∗FilesetName, const uint64_t ∗FilesetId, const ns_ObjHandle_t ∗FilesetHandle, ns_FilesetAttrBits_t FilesetAttrBitsIn, const ns_FilesetAttrs_t ∗FilesetAttrsIn, ns_FilesetAttrBits_t FilesetAttrBitsOut, ns_FilesetAttrs_t ∗FilesetAttrsOut)

  Sets the attributes of a fileset.

11.17.1 Detailed Description

Functions for manipulating filesets.

11.18 api_fsetattr.c File Reference

Functions for setting file attributes.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include "hpss_api.h"
#include "hpss_String_clnt.h"
#include "api_internal.h"
#include "acct_av_lib.h"
#include "hpss_StringUtil.h"
#include <inttypes.h>
```

Functions

- int hpss_FdelFileDigestList (int Fildes, hpss_file_hash_stripe_flags_t Flags)

  Delete file hash digest information.
- int hpss_FileSetAttributes (const char ∗Path, hpss_fileattrbits_t SelFlags, const hpss_fileattr_t ∗AttrIn, hpss_fileattr_t ∗AttrOut)

  Changes the attributes of an object.
- int hpss_FileSetAttributesBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, hpss_fileattrbits_t SelFlags, const hpss_fileattr_t ∗AttrIn, hpss_fileattr_t ∗AttrOut, char ∗Path)

  Change the attributes of an object specified by bitfile object.
- int hpss_FileSetAttributesHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, hpss_fileattrbits_t SelFlags, const hpss_fileattr_t ∗AttrIn, hpss_fileattr_t ∗AttrOut)

  Changes the attributes of an object based upon an object handle and path.
- int hpss_FileSetCOS (const char ∗Path, uint32_t COSId, uint32_t StreamId)

  Change the COS of an file.
- int hpss_FileSetCOSHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t COSId, uint32_t StreamId)

  Change the COS of a file.
- int hpss_FsetFileDigest (int Fildes, const hpss_file_hash_digest_t ∗Digest)

  Sets single stripe file's hash digest information.
- int hpss_FsetFileDigestList (int Fildes, uint64_t StripeLength, const hpss_file_hash_digest_list_t ∗DigestList)

  Sets file hash digest information.
- int hpss_SetAttrHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint64_t SelFlagsIn, const hpss_vattr_t ∗AttrIn, uint64_t ∗SelFlagsOut, hpss_vattr_t ∗AttrOut)

  Updates attribute values for a file or directory.
- int hpss_SetFileNotrunc (const char ∗Path, notrunc_flag_t Flag)

  Sets or clears a file's NOTRUNC_FINAL_SEG flag.

11.18.1 Detailed Description

Functions for setting file attributes.

11.19 api_getcwd.c File Reference

Functions for retrieving the current working directory.

```
#include <sys/param.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Getcwd (char ∗Buf, size_t Size)

  Retrieves the current working directory.
- int hpss_GetThreadUcred (sec_cred_t ∗RetUcred)

  Retrieves the user credentials for the current thread.

11.19.1 Detailed Description

Functions for retrieving the current working directory.

11.20 api_gettrash.c File Reference

Functions for finding HPSS trashcans.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "u_signed64.h"
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_GetTrashcan (const char ∗Path, char ∗TrashcanBuf, int TrashcanBufLength, ns_ObjHandle_t ∗FilesetRoot)

  Locate the trashcan for a provided path.
- int hpss_GetTrashcanHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, char ∗TrashcanBuf, int TrashcanBufLength, ns_ObjHandle_t ∗FilesetRoot)

  Locate the trashcan for a provided handle/path.

11.20.1 Detailed Description

Functions for finding HPSS trashcans.

11.21 api_ids.c File Reference

Functions for mapping between names and ids.

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "hpss_sec_clnt.h"
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_ConvertIdsToNames (int32_t NumEntries, api_namespec_t ∗Specs)
  Converts user or group ids to names.
- int hpss_ConvertNamesToIds (int32_t NumEntries, api_namespec_t ∗Specs)
  Converts user or group names to ids.
- int32_t hpss_GetGID (gid_t ∗GID)
  Get the group id for the calling thread.
- int32_t hpss_GetUID (uid_t ∗UID)
  Get the user id for the calling thread.
- int hpss_SetGID (uint32_t NewCurGid)
  Set the group id for the calling thread.
- int hpss_SiteIdToName (const hpss_id_t ∗SiteId, char ∗SiteName)
  Convert Site ID to Site Name.
- int hpss_SiteNameToId (const char ∗SiteName, hpss_id_t ∗SiteId)
  Convert Site Name to Site ID.

11.21.1 Detailed Description

Functions for mapping between names and ids.

11.21.2 Function Documentation

11.21.2.1 int32_t hpss_GetUID (uid_t ∗ *UID*)

Get the user id for the calling thread.

client_api

The 'hpss_GetUID' function gets the current UID for the calling thread.

Return values

0	Success
<0	Error from API_ClientAPIInit().

11.22 api_init.c File Reference

Functions for initializing the Client API.

```
#include <inttypes.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <unistd.h>
#include <stdarg.h>
#include <pwd.h>
#include <grp.h>
#include "hpss_pthread.h"
#include "hpss_types.h"
#include "hpss_gssapi.h"
#include "hpss_sec_clnt.h"
#include "hpss_api.h"
#include "hpss_connmgr.h"
#include "hpss_server_types.h"
#include "hpss_mech.h"
#include "api_internal.h"
#include "core_interface.h"
#include "acct_av_lib.h"
#include "hpss_Getenv.h"
#include "hpss_String.h"
#include "hpss_uuid.h"
#include "hpss_thread_safe.h"
#include "hpss_InfraLog.h"
```

Functions

- int API_ClientAPIInit (apithrdstate_t ∗∗ThreadContext)

  Initialize threads if not already done.
- void API_DebugPrintError (FILE ∗File, const hpss_reqid_t ∗RequestID, const char ∗Format,...)

  Prints formatted data to a stream.
- void API_DebugPrintRequest (FILE ∗File, const hpss_reqid_t ∗RequestID, const char ∗Format,...)

  Prints formatted data to a stream.
- void API_DebugPrintTrace (FILE ∗File, const hpss_reqid_t ∗RequestID, const char ∗Format,...)

  Prints formatted data to a stream.
- int API_GetDataThreadStackSize (void)

  Retrieves the data thread stack size.
- int API_GetServerIDForSubsystem (uint32_t SubsystemID, hpss_reqid_t RequestID, hpss_srvr_id_t ∗RetID)

  Retrieves the server id of the core server of a given subsystem.
- int API_GetServerIDFromObjHandle (const ns_ObjHandle_t ∗ObjHandle, uint32_t ∗ServerType, hpss_srvr_-id_t ∗ID)

Retrieve the server id of the core server of a given object.

- int API_GrabConn (const char ∗FunctionName, hpss_reqid_t RequestId, int ServerType, const hpss_srvr_id-_t ∗ServerID, apicsepinfo_t ∗CSEPInfo, hpss_ConnMgrEntry_t ∗∗Conn)

  Obtains control of a new or existing server connection.
- int API_InitRootHandle (apithrdstate_t ∗ThreadContext, hpss_reqid_t RequestID, const ns_ObjHandle_-t ∗∗HandlePtr, u_signed64 ∗FilesetId, hpss_Attrs_t ∗Attrs)

  Initializes and/or returns the NS root handle.
- int API_ReleaseConn (const char ∗FunctionName, hpss_reqid_t RequestId, hpss_ConnMgrEntry_t ∗Conn, apicsepinfo_t ∗CSEPInfo, int32_t Error, int32_t RpcErr)

  Releases control of a connection.
- void API_reset_ResourceLog ()

  Reset the resource log info.
- void API_SetAcctID (acct_rec_t AcctID)

  Sets the current account ID in the global thread state.
- void API_SetCurrentGID (uint32_t NewGID)

  Sets the current group ID in the global thread state.
- void API_SetCwd (const ns_ObjHandle_t ∗CwdPtr, const acct_rec_t ∗AcctCode, const unsigned32 ∗COSId, const u_signed64 ∗FilesetId, const hpss_Attrs_t ∗Attrs)

  Sets the current working directory in the global thread state.
- void API_SetThreadCwd (apithrdstate_t ∗ThreadContext, const ns_ObjHandle_t ∗ObjHandle, const u_-signed64 ∗FilesetId, const acct_rec_t ∗Account, const unsigned32 ∗COSId, const hpss_Attrs_t ∗Attrs)

  Set attributes of a thread cwd state.
- void API_SetUmask (mode_t Umask)

  Sets the user mask in the global thread state.
- int hpss_ClientAPIInit (void)

  Initialize the Client API for the current thread.
- void hpss_ClientAPIReset ()

  Resets the client API state.
- int hpss_GetConfiguration (api_config_t ∗ConfigOut)

  Retrieves the current values used for default configurations.
- int hpss_LoadDefaultThreadState (uid_t UserID, mode_t Umask, const char ∗ClientFullName)

  Allows some clients to manipulate the global state of a thread.
- int hpss_LoadThreadState (uid_t UserID, mode_t Umask, const char ∗ClientFullName)

  Allows some clients to manipulate the local state of a thread.
- int hpss_PingCore (const hpss_srvr_id_t ∗CoreServerID, uint32_t ∗RecvSecs, uint32_t ∗RecvUSecs)

  Pings the Core Server.
- void hpss_PurgeLoginCred (void)

  Purges an HPSS login credential.
- int hpss_SetAPILogLevel (int LogLevel)

  Modifies the Client API log level.
- int hpss_SetAPILogPath (const char ∗LogPath)

  Modifies the Client API log path.
- void hpss_SetApplicationName (const char ∗App)

  Modifies the Client API application name.
- int hpss_SetAppLoginCred (const char ∗AppName, const char ∗PrincipalName, hpss_authn_mech_t Mechanism, hpss_rpc_cred_type_t CredType, hpss_rpc_auth_type_t AuthType, const void ∗Authenticator)

  Sets an HPSS login credential.
- int hpss_SetAuditInfo (hpss_sockaddr_t EndUserHostAddr)

  Sets audit information for threads.
- int hpss_SetConfiguration (const api_config_t ∗ConfigIn)

  Sets the values used for the API configuration settings.

- int hpss_SetLoginCred (const char ∗PrincipalName, hpss_authn_mech_t Mechanism, hpss_rpc_cred_type_t CredType, hpss_rpc_auth_type_t AuthType, const void ∗Authenticator)

  Sets an HPSS login credential.
- int hpss_ThreadCleanUp ()

  Cleans up a thread's context.

Variables

- int API_TransferType = 1

11.22.1 Detailed Description

Functions for initializing the Client API.

11.22.2 Function Documentation

11.22.2.1 int API_ClientAPIInit (apithrdstate_t ∗∗ *ThreadContext*)

Initialize threads if not already done.

If the thread's state has already been initialized just return a pointer to the thread's specific state. Otherwise initialize and connect to all necessary servers and then return the new thread specific state.

Parameters

`out`	*ThreadContext*	state of thread

Return values

0	No error. Thread specific state is valid.

11.22.2.2 void API_DebugPrintError (FILE ∗ *File*, const hpss_reqid_t ∗ *RequestID*, const char ∗ *Format*, ...)

Prints formatted data to a stream.

These routine prints formatted data to a stream. The arguments follow format of the fprintf routine. A timestamp is inserted in front of all text to be output in the format string.

Parameters

`in`	*File*	Pointer to a FILE structure
`in`	*RequestID*	Pointer to a request id
`in`	*Format*	Format-control string

Return values

None.	

11.22.2.3 void API_DebugPrintRequest (FILE ∗ *File*, const hpss_reqid_t ∗ *RequestID*, const char ∗ *Format*, ...)

Prints formatted data to a stream.

These routine prints formatted data to a stream. The arguments follow format of the fprintf routine. A timestamp is inserted in front of all text to be output in the format string.

Parameters

`in`	*File*	Pointer to a FILE structure
`in`	*RequestID*	Pointer to a request id
`in`	*Format*	Format-control string

Return values

None.	

11.22.2.4 void API_DebugPrintTrace (FILE ∗ *File,* const hpss_reqid_t ∗ *RequestID,* const char ∗ *Format, ...*)

Prints formatted data to a stream.

These routine prints formatted data to a stream. The arguments follow format of the fprintf routine. A timestamp is inserted in front of all text to be output in the format string.

Parameters

`in`	*File*	Pointer to a FILE structure
`in`	*RequestID*	Pointer to a request id
`in`	*Format*	Format-control string

Return values

None.	

11.22.2.5 int API_GetServerIDFromObjHandle (const ns_ObjHandle_t ∗ *ObjHandle,* uint32_t ∗ *ServerType,* hpss_srvr_id_t ∗ *ID*)

Retrieve the server id of the core server of a given object.

Get the server id of the core server who controls the object referenced by the specified object handle. If the handle is null, or is composed entirely of zeroes, return the server id of the root core server. Return as ServerType either "core server" or "root core server", as appropriate.

Parameters

`in`	*ObjHandle*	input object handle
`out`	*ServerType*	server type
`out`	*ID*	server id

Return values

0	No error
Other	Value from hpss_LocateRootCS()

11.22.2.6 int API_GrabConn (const char ∗ *FunctionName,* hpss_reqid_t *RequestId,* int *ServerType,* const hpss_srvr_id_t ∗ *ServerID,* apicsepinfo_t ∗ *CSEPInfo,* hpss_ConnMgrEntry_t ∗∗ *Conn*)

Obtains control of a new or existing server connection.

This routine will obtain control of a new or existing connection to the server specified by the Server ID and Server-Type input parameters.

Parameters

`in`	*FunctionName*	calling function name
`in`	*RequestId*	request id
`in`	*ServerType*	Type of the server
`in`	*ServerID*	Server to get connect for
`in`	*CSEPInfo*	core server EP info
`out`	*Conn*	connection entry

Return values

-EFAULT	Invalid connection input
-EINVAL	Server's location is not local and cross-cells disabled
-ESYSTEM	Build problem with the library
0	No error
Other	Error returned by hpss_LocateServersByType()or hpss_ConnMgrGrabConn()

11.22.2.7 int API_InitRootHandle (apithrdstate_t ∗ *ThreadContext,* hpss_reqid_t *RequestID,* const ns_ObjHandle_t ∗∗ *HandlePtr,* u_signed64 ∗ *FilesetId,* hpss_Attrs_t ∗ *Attrs*)

Initializes and/or returns the NS root handle.

This routine is called to initialize and/or return the NS root handle.

Parameters

`in`	*ThreadContext*	thread context
`in`	*RequestID*	request id
`in,out`	*HandlePtr*	root handle pointer
`in,out`	*FilesetId*	root fileset id

Return values

0	Success
HandlePtr	pointer to the NS root handle
Other	Another HPSS errno indicating the nature of the error

11.22.2.8 int API_ReleaseConn (const char ∗ *FunctionName,* hpss_reqid_t *RequestId,* hpss_ConnMgrEntry_t ∗ *Conn,* apicsepinfo_t ∗ *CSEPInfo,* int32_t *Error,* int32_t *RpcErr*)

Releases control of a connection.

This routine will release control of a previous obtain connection. If the RPC or return error status indicates a communication occured action will be taken to repair the connection before returning control to the caller.

Parameters

`in`	*FunctionName*	calling function name
`in`	*RequestId*	request id
`in`	*Conn*	connection entry
`in`	*CSEPInfo*	core server EP info
`in`	*Error*	Error from the server

in	*RpcErr*	Error from the RPC

Return values

-EBADCONN	The connection went bad
0	No error
Other	Error returned by hpss_ConnMgrReleaseConn()

11.22.2.9 void API_SetAcctID (acct_rec_t *AcctID*)

Sets the current account ID in the global thread state.

This routine will set the current account ID in the global thread state

Parameters

in	*AcctID*	account ID user mask

Return values

None.	

11.22.2.10 void API_SetCurrentGID (uint32_t *NewGID*)

Sets the current group ID in the global thread state.

This routine will set the current Group ID in the global thread state

Parameters

in	*NewGID*	new current Gid

Return values

None.	

11.22.2.11 void API_SetCwd (const ns_ObjHandle_t ∗ *CwdPtr,* const acct_rec_t ∗ *AcctCode,* const unsigned32 ∗ *COSId,* const u_signed64 ∗ *FilesetId,* const hpss_Attrs_t ∗ *Attrs*)

Sets the current working directory in the global thread state.

Parameters

in	*CwdPtr*	ptr to directory handle
in	*AcctCode*	ptr to account code
in	*COSId*	ptr to class of service id
in	*FilesetId*	Fileset id for cwd
in	*Attrs*	ptr to directory attributes

This routine will set the current working directory in the global thread state

Return values

None.	

11.22.2.12 void API_SetThreadCwd (apithrdstate_t ∗ *ThreadContext*, const ns_ObjHandle_t ∗ *ObjHandle*, const u_signed64 ∗ *FilesetId*, const acct_rec_t ∗ *Account*, const unsigned32 ∗ *COSId*, const hpss_Attrs_t ∗ *Attrs*) [inline]

Set attributes of a thread cwd state.

This routine is called set attributes of the thread's cwd state.

Parameters

in,out	*ThreadContext*	Thread context to modify
in	*ObjHandle*	CWD Object Handle
in	*FilesetId*	Fileset Id for the CWD
in	*Account*	Account Code for the CWD
in	*COSId*	COSId for the CWD
in	*Attrs*	Attributes for CWD

11.22.2.13 void API_SetUmask (mode_t *Umask*)

Sets the user mask in the global thread state.

This routine will set the user mask in the global thread state

Parameters

in	*Umask*	user mask

Return values

None.	

11.22.3 Variable Documentation

11.22.3.1 int API_TransferType = 1

Mover selects the protocol

11.23 api_junction.c File Reference

Functions for manipulating junctions.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_GetJunctions (uint32_t SubsystemID, uint64_t OffsetIn, uint32_t Entries, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_junction_ent_t ∗JentPtr)

  Retrives the junctions that are managed by the Core Server.

- int hpss_JunctionCreate (const char ∗Path, const ns_ObjHandle_t ∗SourceHandle, ns_ObjHandle_t ∗JunctionHandle)

  Creates an HPSS junction.
- int hpss_JunctionCreateHandle (const ns_ObjHandle_t ∗ParentHandle, const char ∗Path, const ns_Obj-Handle_t ∗SourceHandle, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗JunctionHandle)

  Creates an HPSS junction in relation to a fileset handle.
- int hpss_JunctionDelete (const char ∗Path)

  Deletes a junction.
- int hpss_JunctionDeleteHandle (const ns_ObjHandle_t ∗ParentHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Deletes a junction.

11.23.1 Detailed Description

Functions for manipulating junctions.

11.24 api_link.c File Reference

Functions for creating links.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Link (const char ∗Existing, const char ∗New)

  Creates a link to a file.
- int hpss_LinkHandle (const ns_ObjHandle_t ∗ObjHandle, const ns_ObjHandle_t ∗DirHandle, const char ∗New, const sec_cred_t ∗Ucred)

  Creates a link.
- int hpss_LinkHandleParent (const ns_ObjHandle_t ∗SrcDirHandle, const char ∗SrcFile, const ns_ObjHandle-_t ∗DestDirHandle, const char ∗DestFile, const sec_cred_t ∗Ucred)

  Creates a link.

11.24.1 Detailed Description

Functions for creating links.

11.25 api_lookup.c File Reference

Functions for looking up object ids or handles.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
#include "hpss_server_types.h"
```

Functions

- int hpss_GetObjFromHandle (const ns_ObjHandle_t ∗ObjHandle, object_id_hash_t ∗ObjHash)

  Retrieves an object from a handle.
- uint32_t hpss_GetObjType (const ns_ObjHandle_t ∗ObjHandle)

  Retrieves the object type given its handle.
- int hpss_LookupRootCS (const char ∗SiteName, hpss_srvr_id_t ∗ServerId)

  Retrieves the ID for a site's root Core Server.

11.25.1 Detailed Description

Functions for looking up object ids or handles.

11.26 api_lseek.c File Reference

Functions for modifying the offset pointer within a file.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int64_t hpss_Lseek (int Fildes, int64_t Offset, int Whence)

  Sets the file offset for the open file handle.
- int hpss_SetFileOffset (int Fildes, uint64_t OffsetIn, int Whence, int Direction, uint64_t ∗OffsetOut)

  Sets the file offset for an open file handle.

11.26.1 Detailed Description

Functions for modifying the offset pointer within a file.

11.27 api_misc.c File Reference

Useful misc. Client API functions.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h>
#include <pthread.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/select.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include "pdata.h"
#include "hpss_server_types.h"
#include "hpss_api.h"
#include "hpss_netopt.h"
#include "cfg_extensions/hpsscfgx_prototypes.h"
#include "api_internal.h"
#include "hpss_StringUtil.h"
#include "hpss_Getenv.h"
#include "hpss_constants.h"
#include <inttypes.h>
```

Functions

- int API_AllocateRequestIOR (const IOD_t ∗IODPtr, hpss_IOR_t ∗IORPtr)

  Allocates memory for an IOR.
- int API_BuildFileSrcSink (iod_srcsinkdesc_t ∗DescList, filetable_t ∗FTPtr, open_bf_desc_t ∗∗OpenBFDescPtr, srcsinkdesc_t ∗∗RetDescList)

  Builds the bitfile side of a source/sink descriptor list.
- int API_BuildXferSrcSink (iod_srcsinkdesc_t ∗DescList, open_bf_desc_t ∗OpenBFDesc, hpss_reqid_t RequestId, int ∗ListenDesc, int ∗DataDesc, data_addr_t ∗∗DataAddrPtr, uint64_t ∗MemoryLength, srcsinkdesc_t ∗∗RetDescList)

  Builds the transfer side of a source/sink descriptor list.
- void API_CalcBlock32 (const hpss_Attrs_t ∗Attrs, uint32_t ∗Blksize, uint32_t ∗Numblks)

  Calculates the block size and number of blocks necessary for data.
- void API_CalcBlock64 (const hpss_Attrs_t ∗Attrs, uint64_t ∗Blksize, uint64_t ∗Numblks)

  Calculates the block size and number of blocks necessary for data.
- int API_CalcBufOffsetAndSize (uint64_t Offset, int BufSize, const char ∗Buffer, uint64_t MessageOffset, uint64_t MessageLength, char ∗∗RetBufStart, int ∗RetLength)

  Calculates the buffer offset and size for a data transfer.
- void API_CancelDataTask (open_bf_desc_t ∗OpenBFDescPtr, int ∗Flags, pthread_mutex_t ∗MutexPtr)

  Causes a data thread to end its select loop and exit.
- int API_CheckFileDescBounds (const filetable_t ∗Ftptr, int Fildes)

  Check the file descriptor against the file table bounds.
- void API_CleanupInterruptBuffer (open_bf_desc_t ∗OpenBFDescPtr)

  Removes pending activity from a thread control buffer.
- void API_ClearFileTableBusy (filetable_t ∗Ftptr, int Fildes)

  Clear a file table entry marked busy.
- void API_CloseDataConnList (desc_elem_t ∗∗ListPtr)

  Closes connections for a file.
- void API_CloseListenDesc (int Type, int ∗ListenDesc)

  Closes a listen port for a thread.

- int API_CloseThrdCntlDesc (open_bf_desc_t ∗OpenBFDescPtr)

  Closes a thread control pipe to end a data thread.
- void API_ConvertAttrsToVAttrs (const hpss_Attrs_t ∗Attrs, hpss_vattr_t ∗VAttrs)

  Converts HPSS attributes to VFS attributes.
- void API_ConvertAttrToVAttrSelFlags (uint64_t SelFlags, uint64_t ∗RetFlagsPtr)

  Converts an HPSS selection bitmap to a VFS selection bitmap.
- void API_ConvertModeToPosixMode (const hpss_Attrs_t ∗Attrs, mode_t ∗PosixMode)

  Converts an input HPSS mode value to its corresponding POSIX mode value.
- void API_ConvertPosixModeToMode (mode_t PosixMode, hpss_Attrs_t ∗Attrs)

  Converts a POSIX mode value to its equivalent HPSS mode value.
- void API_ConvertPosixTimeToTime (hpss_Attrs_t ∗Attrs, timestamp_sec_t Atime, timestamp_sec_t Mtime, timestamp_sec_t Ctime)

  Converts POSIX time values to corresponding HPSS time values.
- void API_ConvertTimeToPosixTime (const hpss_Attrs_t ∗Attrs, timestamp_sec_t ∗Atime, timestamp_sec_t ∗Mtime, timestamp_sec_t ∗Ctime)

  Converts HPSS time values to corresponding POSIX mode values.
- void API_ConvertVAttrsToAttrs (const hpss_vattr_t ∗VAttrs, hpss_Attrs_t ∗Attrs)

  Converts VFS attributes to HPSS attributes.
- void API_ConvertVAttrToAttrSelFlags (uint64_t VAttrSelFlags, hpss_AttrBits_t ∗RetFlagsPtr)

  Converts a VFS selection bitmap to an HPSS selection bitmap.
- api_desc_node_t ∗ API_DescListAlloc (int Fildes)

  Initialize a descriptor list node.
- void API_DescListClear (int Fildes, hpss_ilist_t ∗FildesLocked)

  Clear a file descriptor as locked in the list.
- void API_DescListFree (hpss_ilist_t ∗FildesLocked)

  Free a descriptor list.
- int API_DescListIsSet (int Fildes, hpss_ilist_t ∗FildesLocked)

  Determine whether a file descriptor is locked in the list.
- void API_DescListSet (int Fildes, hpss_ilist_t ∗FildesLocked)

  Set a file descriptor as locked in the list.
- hpss_ilist_t ∗ API_DescListZero (hpss_ilist_t ∗FildesLocked)

  Initialize the descriptor lock list.
- int API_DivideFilePath (const char ∗PathIn, char ∗PathOut, char ∗FileOut)

  Translates a path to either a file name or a file name and a path.
- int API_ExpandFileTable (filetable_t ∗Ftptr)

  Expand the size of the file table.
- uint64_t API_FileTableSize (const filetable_t ∗Ftptr)

  Get the current file table size.
- void API_FillFreeList (hpss_ilist_t ∗List, uint64_t StartDes, uint64_t EndDes)

  Fill entries to the free list in a range [start,end].
- void API_FreeIODPtrs (IOD_t ∗IODPtr)

  Frees memory that was allocated to an IOD.
- void API_FreeIOR (hpss_IOR_t ∗IORPtr)

  Frees memory that was allocated to an IOR.
- int API_FreeRequestIOR (const IOD_t ∗IODPtr, IOR_t ∗IORPtr)

  Frees memory that was allocated to an IOR.
- int API_GetAllSubsystems (ls_map_array_t ∗∗ls_map_array)

  Retrieve a map of all subsystems.
- int API_GetAppliedPath (const char ∗Path, char ∗∗Newpath, apithrdstate_t ∗Threadcontext)

  Applies the root directory to the given path and returns the new path.

- int API_GetClientAddr (const iod_srcsinkdesc_t ∗SrcSinkPtr, const iod_srcsinkreply_t ∗SrcSinkReplyPtr, uint64_t Offset, iod_address_t ∗∗RetAddr, uint64_t ∗RetLength, iod_srcsinkdesc_t ∗∗RetDescPtr, iod_-srcsinkreply_t ∗∗RetReplyPtr)

    Retrieves the client address for a data transfer.
- hpss_reqid_t API_GetNextIORequestID (apithrdstate_t ∗ThreadContext)

    Retrieves the next request id that will be used for I/O and generates a new one.
- void API_GetObjectFilesetId (apithrdstate_t ∗ThreadContext, hpss_reqid_t RequestId, const sec_cred_-t ∗Ucred, int ChaseFlags, const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, api_cwd_stack_t ∗CwdStack, const u_signed64 ∗FilesetIdHint, u_signed64 ∗FilesetId)

    Retrieve an object's fileset id.
- uint64_t API_GetOpenTableSlot (filetable_t ∗Ftptr)

    Find an open file table spot.
- hpss_reqid_t API_GetUniqueRequestID ()

    Retrieves a unique request id.
- char ∗ API_HexDecode (const char ∗buffer, int ∗len)

    Decode a hex buffer as binary into a buffer string.
- char ∗ API_HexEncode (const char ∗buffer, int buflen)

    Encode a digest buffer into a hex string.
- int API_InitializeServerIOD (const hpss_IOD_t ∗IODPtr, IOD_t ∗RetIODPtr)

    Initiliazes a server IOD.
- void API_InsertInFreeList (hpss_ilist_t ∗List, uint64_t Des)

    Put an entry back into the free list.
- int API_InterruptDataTask (open_bf_desc_t ∗OpenBFDescPtr)

    Writes data to a piped read descriptor.
- int32_t API_LockCondition (api_condition_variable_t ∗Lock, const char ∗Function)

    Uses a condition variable to lock access to data.
- void API_LockMutex (pthread_mutex_t ∗MutexPtr)

    Locks a pthread mutex.
- int API_net_bind (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

    Bind wrapper for busy retry handling.
- int API_Net_CreateAddr_INADDR_ANY (hpss_sockaddr_t ∗SockAddr, int Port, hpss_reqid_t ReqId)

    Creates a socket address for listen / bind calls.
- void API_ObjHandleToString (const ns_ObjHandle_t ∗Handle, char ∗Buffer)

    Displays an NS object handle in text form.
- int API_OpenListenDesc (int Type, int ∗ListenDesc, data_addr_t ∗ListenAddr)

    Opens a list port for a thread.
- int API_OpenThrdCntlDesc (open_bf_desc_t ∗OpenBFDescPtr)

    Creates a thread control pipe for ending a data thread.
- int API_PerformRetry (retry_cb_t ∗RetryCB, int Error)

    Retries a call to an API.
- int32_t API_RemoveCachedTrash (hpss_reqid_t RequestID, const sec_cred_t ∗Ucred, const u_signed64 ∗F-SID, int OptionFlags)

    Remove a cached trashcan entry.
- int API_RetrieveTrashcanHandle (apithrdstate_t ∗ThreadContext, hpss_reqid_t RequestID, const sec_cred-_t ∗Ucred, const ns_ObjHandle_t ∗ParentHandle, const u_signed64 ∗FSId, api_cwd_stack_t ∗CwdStack, ns_ObjHandle_t ∗∗TrashcanContainerHandle, ns_ObjHandle_t ∗∗TrashcanHandle, int ∗OptionFlags)

    Find the appropriate trashcan to use based upon the fileset, parent handle, and user.
- int API_SetFileTableBusy (filetable_t ∗Ftptr, int Fildes, int FlagChecks)

    Mark a file table entry as busy.
- void API_SetLastHPSSErrno (apithrdstate_t ∗Threadcontext, int32_t HPSSErrno, const char ∗Func, hpss_-reqid_t RequestId)

    Sets the current HPSS errno state.

- void API_SocketSetOptions (int Socket, hpss_sockaddr_t ∗SaddrPtr, unsigned int Options)

  Sets the API socket options.
- int API_TranslateError (int32_t Error, retry_cb_t ∗RetryCB)

  Translates an HPSS error into a POSIX error.
- int32_t API_UnlockCondition (api_condition_variable_t ∗Lock, const char ∗Function)

  Unlocks data controlled by a condition variable.
- void API_UnlockFileTableEntries (iod_srcsinkdesc_t ∗SrcSinkPtr, filetable_t ∗FileTabPtr, hpss_ilist_t ∗Fildes-Locked, int NumLocked, int LockTable)

  Unlocks entries in a client file table if previously marked busy.
- void API_UnlockMutex (pthread_mutex_t ∗MutexPtr)

  Unlocks a pthread mutex.
- int hpss_AbortIOByMoverId (int32_t SubsystemId, hpss_srvr_id_t MoverId)

  Abort all I/O requests for the specified mover ID.
- int hpss_AbortIOByRequestId (int32_t SubsystemId, hpss_reqid_t RequestIdToAbort, sec_cred_t ∗Ucred)

  Abort a specific I/O request.
- int hpss_AbortIOByRequestMatch (int32_t SubsystemId, char ∗PartialReqString, sec_cred_t ∗Ucred)

  Abort an I/O request that matches the request string.
- int hpss_AbortIOByUserId (int32_t SubsystemId, sec_cred_t ∗Ucred, uint32_t ∗RequestsAborted, uint32_t ∗RequestsAlreadyAborted, uint32_t ∗RequestsFound)

  Abort all I/O requests for the specified user.
- void hpss_ClearLastHPSSErrno ()

  Clears the current HPSS errno state.
- int hpss_GetAllSubsystems (ls_map_array_t ∗∗ls_map_array)

  Retrieve a map of all subsystems.
- hpss_read_queue_list_t hpss_GetIntersectedList (hpss_read_queue_list_t ∗List, hpss_read_queue_list_t ∗Partial)

  Return a list whose content is all items in the list which do appear in the sublist.
- hpss_errno_state_t hpss_GetLastHPSSErrno ()

  Retrieves the error state of the last HPSS error.
- hpss_reqid_t hpss_GetNextIORequestID ()

  Retrieves the next request id that will be used for I/O.
- void hpss_HashFlagsToString (unsigned16 flags, char ∗buf, int len)

  Returns a string representation of the provided hash flags.
- hpss_read_queue_list_t hpss_RemoveIntersectionFromList (hpss_read_queue_list_t ∗List, hpss_read_queue_list_t ∗Partial)

  Return a list whose content is all items in the list which do not appear in the sublist.
- int32_t hpss_SetNextIORequestID (hpss_reqid_t ∗RequestId)

  Set the request ID to be used for the next I/O in this thread. This should be a request ID generated from hpss_Get-UniqueRequestID or from hpss_ReadQueueAdd.

11.27.1 Detailed Description

Useful misc. Client API functions.

11.27.2 Function Documentation

11.27.2.1 int API_AllocateRequestIOR (const IOD_t ∗ *IODPtr,* hpss_IOR_t ∗ *IORPtr*)

Allocates memory for an IOR.

This routine allocates memory required for the IOR that will be returned in reply to the request, specified by the IOD pointed to by IODPtr. IORPtr must contain a skeleton IOR, to which the request specific memory addresses will be anchored.

Parameters

`in`	*IODPtr*	pointer to IOD describing request
`in`	*IORPtr*	pointer to skeleton IOR

Return values

0	Success
-EINVAL	Request type is unkown
other	Error in allocating memory

11.27.2.2 int API_BuildFileSrcSink (iod_srcsinkdesc_t ∗ *DescList*, filetable_t ∗ *FTPtr*, open_bf_desc_t ∗∗ *OpenBFDescPtr*, srcsinkdesc_t ∗∗ *RetDescList*)

Builds the bitfile side of a source/sink descriptor list.

Parameters

`in`	*DescList*	client IOD descriptor list
`in`	*FTPtr*	pointer to the file table
`in`	*OpenBFDescPtr*	open bitfile descriptor
`in,out`	*RetDescList*	server IOD descriptor list

This routine is called to build the bitfile side for an HPSS IOD source/sink descriptor list.

Return values

0	- on success
-ENOMEM	- if memory allocation fails

11.27.2.3 int API_BuildXferSrcSink (iod_srcsinkdesc_t ∗ *DescList*, open_bf_desc_t ∗ *OpenBFDesc*, hpss_reqid_t *RequestId*, int ∗ *ListenDesc*, int ∗ *DataDesc*, data_addr_t ∗∗ *DataAddrPtr*, uint64_t ∗ *MemoryLength*, srcsinkdesc_t ∗∗ *RetDescList*)

Builds the transfer side of a source/sink descriptor list.

This routine is called to build the data transmit side for an HPSS IOD source/sink descriptor list.

Parameters

`in`	*DescList*	Client IOD descriptor list
`in`	*OpenBFDesc*	Open bitfile descriptor
`in`	*RequestId*	IOD request number
`in,out`	*ListenDesc*	Listen socket descriptor
`out`	*DataDesc*	Data I/O socket descriptor
`out`	*DataAddrPtr*	Data address information
`out`	*MemoryLength*	Bytes of memory to use
`in,out`	*RetDescList*	Server IOD descriptor list

Return values

0	on success
-ENOMEM	if memory allocation fails
-EIO	if descriptor allocation fails

11.27.2.4 int API_CalcBufOffsetAndSize (uint64_t *Offset,* int *BufSize,* const char ∗ *Buffer,* uint64_t *MessageOffset,* uint64_t *MessageLength,* char ∗∗ *RetBufStart,* int ∗ *RetLength*)

Calculates the buffer offset and size for a data transfer.

The 'API_CalcBufOffsetAndSize' function verifies that an inbound data transfer message refers to data within the current buffer, based upon the client's Offset and Buffer Size, and returns the relative buffer address and length to be used for the actual data transfer if the message boundaries are contained within the caller's buffer boundaries.

Parameters

in	*Offset*	Client's relative offset into the transfer
in	*BufSize*	Client's buffer size
in	*Buffer*	Client's buffer
in	*MessageOffset*	Message transfer offset
in	*MessageLength*	Message bytes to transfer
in	*RetBufStart*	Where in Buffer to start the data traansfer
out	*RetLength*	How many bytes to transfer

Return values

-EINVAL	validity checks on the message failed.
0	message passed all checks.

Note

This code is currently used only for SAN3P transfers via hpss_Read/hpss_Write

11.27.2.5 void API_CancelDataTask (open_bf_desc_t ∗ *OpenBFDescPtr,* int ∗ *Flags,* pthread_mutex_t ∗ *MutexPtr*)

Causes a data thread to end its select loop and exit.

This function will cause the data thread to end its select loop and exit.

Parameters

in	*OpenBFDescPtr*	Object Handle pointer
in	*Flags*	Data Thread Flags
in	*MutexPtr*	Data Thread Mutex

Note

Assumptions:

This function relies on the pipe created by hpss_Open and logic in the data thread of each I/O function (hpss_Write, hpss_Read, hpss_Writelist, hpss_Readlist) that interprets activity on the read portion of that pipe as a termination condition.

It is assumed that the data thread mutex is locked when this function is called.

Note

Utilized instead of pthread_cancel because in some instances, even though the pthread_cancel would end the thread successfully, the subsequent call to pthread_join would cause a crash.

11.27.2.6 int API_CheckFileDescBounds (const filetable_t ∗ *Ftptr,* int *Fildes*)

Check the file descriptor against the file table bounds.

Parameters

in	*Ftptr*	File table
in	*Fildes*	File descriptor

Return values

HPSS_ERANGE	File descriptor is out of range
HPSS_E_NOERROR	File descriptor is in range

11.27.2.7 void API_CleanupInterruptBuffer (open_bf_desc_t ∗ *OpenBFDescPtr*)

Removes pending activity from a thread control buffer.

Will remove still pending activity from the thread control buffer.

Parameters

in	*OpenBFDescPtr*	Object Handle pointer

Note

This function assumes that the write descriptor is not being written to while this cleanup is occuring. That situation could easily lead to the Client API stalling.
This function is required because the write portion of the FD may be triggered multiple times while the read portion is triggered once, or an I/O operation may terminate before the descriptor is read. Since the descriptors are asynchronous, those messages are still waiting to be read. So, every time we finish, we are forced to clean out the activity that was created.

11.27.2.8 void API_ClearFileTableBusy (filetable_t ∗ *Ftptr,* int *Fildes*)

Clear a file table entry marked busy.

Clear the ENTRY_BUSY flag for an open file descriptor

Parameters

in	*Ftptr*	File table
in	*Fildes*	Open file descriptor

11.27.2.9 void API_CloseDataConnList (desc_elem_t ∗∗ *ListPtr*)

Closes connections for a file.

Parameters

in	*ListPtr*	Pointer to the list head

The API_CloseDataConnList routine is called to close the connections for a given file.

11.27.2.10 void API_CloseListenDesc (int *Type,* int ∗ *ListenDesc*)

Closes a listen port for a thread.

Parameters

`in`	*Type*	Type of descriptor
`in,out`	*ListenDesc*	Descriptor to close

The API_CloseListenDesc routine is called by a thread when that thread wishes to close a listen port.

Return values

0	NoError
negative	error closing the descriptor

Note

ListenDesc will be set to API_INVALID_FD after close.

11.27.2.11 int API_CloseThrdCntlDesc (open_bf_desc_t * *OpenBFDescPtr*)

Closes a thread control pipe to end a data thread.

Close the thread control pipe used to end a data thread

Parameters

`in`	*OpenBFDescPtr*	Open Bitfile Descriptor

Return values

0	Success.
Other	Error returned from close()

11.27.2.12 void API_ConvertAttrsToVAttrs (const hpss_Attrs_t * *Attrs,* hpss_vattr_t * *VAttrs*)

Converts HPSS attributes to VFS attributes.

This routine translates HPSS attributes to vfs attributes.

Parameters

`in`	*Attrs*	Ptr to HPSS attributes
`out`	*VAttrs*	Ptr to VFS attributes

Return values

None.	

11.27.2.13 void API_ConvertAttrToVAttrSelFlags (uint64_t *SelFlags,* uint64_t * *RetFlagsPtr*)

Converts an HPSS selection bitmap to a VFS selection bitmap.

This routine translates a HPSS selection bitmap to a vfs selection bitmap.

Parameters

in	*SelFlags*	HPSS selection flags
out	*RetFlagsPtr*	Ptr to converted flags

Return values

None.	

11.27.2.14 void API_ConvertVAttrsToAttrs (const hpss_vattr_t ∗ *VAttrs,* hpss_Attrs_t ∗ *Attrs*)

Converts VFS attributes to HPSS attributes.

This routine translates vfs attributes to HPSS attributes.

Parameters

in	*VAttrs*	Ptr to VFS attributes
out	*Attrs*	Ptr to HPSS attributes

Return values

None	

11.27.2.15 void API_ConvertVAttrToAttrSelFlags (uint64_t *VAttrSelFlags,* hpss_AttrBits_t ∗ *RetFlagsPtr*)

Converts a VFS selection bitmap to an HPSS selection bitmap.

This routine translates a vfs selection bitmap to a HPSS attribute selection bitmap.

Parameters

in	*VAttrSelFlags*	NS selection flags
out	*RetFlagsPtr*	Ptr to converted flags

Return values

None.	

11.27.2.16 api_desc_node_t∗ API_DescListAlloc (int *Fildes*)

Initialize a descriptor list node.

Parameters

in	*Fildes*	File descriptor

Returns

A list node containing the provided file descriptor, or NULL

11.27.2.17 void API_DescListClear (int *Fildes,* hpss_ilist_t ∗ *FildesLocked*)

Clear a file descriptor as locked in the list.

Parameters

in	*Fildes*	File descriptor
in,out	*FildesLocked*	Locked file descriptor list

11.27.2.18 void API_DescListFree (hpss_ilist_t ∗ *FildesLocked*)

Free a descriptor list.

Parameters

in	*FildesLocked*	Descriptor list to free

11.27.2.19 int API_DescListIsSet (int *Fildes,* hpss_ilist_t ∗ *FildesLocked*)

Determine whether a file descriptor is locked in the list.

Parameters

in	*Fildes*	File descriptor
in,out	*FildesLocked*	Locked file descriptor list

11.27.2.20 void API_DescListSet (int *Fildes,* hpss_ilist_t ∗ *FildesLocked*)

Set a file descriptor as locked in the list.

Parameters

in	*Fildes*	File descriptor
in,out	*FildesLocked*	Locked file descriptor list

11.27.2.21 hpss_ilist_t∗ API_DescListZero (hpss_ilist_t ∗ *FildesLocked*)

Initialize the descriptor lock list.

Parameters

in,out	*FildesLocked*	Locked file descriptor list

11.27.2.22 int API_DivideFilePath (const char ∗ *PathIn,* char ∗ *PathOut,* char ∗ *FileOut*)

Translates a path to either a file name or a file name and a path.

The API_DivideFilePath routine is called to translate a path to either a file name or a file name and directory path. 'PathOut' and 'FileOut' should be allocated by the caller (size of HPSS_MAX_PATH_NAME). If 'PathOut' is not needed, the null string will be assigned. If PathIn is a NULL pointer then null strings will be returned for 'PathOut' and 'FileOut'.

Parameters

in	*PathIn*	path and file
out	*PathOut*	returned path
out	*FileOut*	returned file

Return values

-ENAMETOOLONG	The input path name is greater than or equal to HPSS_MAX_PATH_NAME.

Note

Assumptions: PathOut and FileOut are valid pointers to string of size $<$ HPSS_MAX_PATH_NAME.

11.27.2.23 int API_ExpandFileTable (filetable_t * *Ftptr*)

Expand the size of the file table.

Doubles the size of the file table, up to the maximum open file limit. Upon error, the file table is unchanged.

Parameters

in,out	*Ftptr*	File table

Return values

HPSS_E_NOERROR	Expansion completed successfully
HPSS_EMFILE	Too many open files
Other	Other error expanding the open file table

11.27.2.24 uint64_t API_FileTableSize (const filetable_t * *Ftptr*)

Get the current file table size.

Parameters

in	*Ftptr*	File table

Returns

The size of the file table

11.27.2.25 void API_FillFreeList (hpss_ilist_t * *List,* uint64_t *StartDes,* uint64_t *EndDes*)

Fill entries to the free list in a range [start,end].

Parameters

in	*List*	List to add to
in	*StartDes*	Start descriptor number
in	*EndDes*	End descriptor number

Note

The range includes all integers from StartDes to EndDes-1

11.27.2.26 void API_FreeIODPtrs (IOD_t * *IODPtr*)

Frees memory that was allocated to an IOD.

Parameters

in	*IODPtr*	Pointer to IOD describing request

This routine frees memory that was allocated to the IOD.

11.27.2.27 void API_FreeIOR (hpss_IOR_t * *IORPtr*)

Frees memory that was allocated to an IOR.

This routine frees memory that was allocated for a client IOR.

Parameters

in,out	*IORPtr*	pointer to IOR to free

11.27.2.28 int API_FreeRequestIOR (const IOD_t * *IODPtr,* IOR_t * *IORPtr*)

Frees memory that was allocated to an IOR.

This routine frees memory that was allocated to the IOR by AllocateRequestIOR. Actions vary depending on what the request type specified in the IOD.

Parameters

in	*IODPtr*	pointer to IOD describing request
in	*IORPtr*	pointer to IOR to free

Return values

0	Success
-EFAULT	Invalid pointer encountered

11.27.2.29 int API_GetAllSubsystems (ls_map_array_t ** *ls_map_array*)

Retrieve a map of all subsystems.

Parameters

out	*ls_map_array*	Location Map Array

This function finds all subsystems for the current site.

Return values

0	Success
HPSS_ENOENT	No subsystems were returned.
Other	Error communicating with the location server

Note

The caller is responsible for freeing the returned map.

11.27.2.30 int API_GetAppliedPath (const char * *Path,* char ** *Newpath,* apithrdstate_t * *Threadcontext*)

Applies the root directory to the given path and returns the new path.

Applies the Root Directory to the given path if necessary and returns the string which contains the new path.

Parameters

in	*Path*	Path
out	*Newpath*	New Path
in,out	*Threadcontext*	Thread Context

Return values

0	XQuery does not match any invalid components
-ENOMEM	No memory for new string buffer
Other	Error from API_ApplyRootDir

11.27.2.31 int API_GetClientAddr (const iod_srcsinkdesc_t ∗ *SrcSinkPtr,* const iod_srcsinkreply_t ∗ *SrcSinkReplyPtr,* uint64_t *Offset,* iod_address_t ∗∗ *RetAddr,* uint64_t ∗ *RetLength,* iod_srcsinkdesc_t ∗∗ *RetDescPtr,* iod_srcsinkreply_t ∗∗ *RetReplyPtr*)

Retrieves the client address for a data transfer.

This routine determines the client address which is associated with the current offset within the transfer, as well as the number of subsequent bytes which also correspond to that address. Also returned are pointers to the source/sink descriptor and reply structures which correspond the returned address.

Parameters

in	*SrcSinkPtr*	initial src/sink descriptor
in	*SrcSinkReplyPtr*	initial src/sink reply
in	*Offset*	offset within transfer
out	*RetAddr*	returned client length
out	*RetLength*	returned length
out	*RetDescPtr*	returned descriptor ptr
out	*RetReplyPtr*	returned reply ptr

Return values

0	Success
-EFAULT	Could not find client address

11.27.2.32 hpss_reqid_t API_GetNextIORequestID (apithrdstate_t ∗ *ThreadContext*)

Retrieves the next request id that will be used for I/O and generates a new one.

Gets the next request id and replaces it with a new one

Returns

This function returns the next request id to use for I/O

11.27.2.33 void API_GetObjectFilesetId (apithrdstate_t ∗ *ThreadContext,* hpss_reqid_t *RequestId,* const sec_cred_t ∗ *Ucred,* int *ChaseFlags,* const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* api_cwd_stack_t ∗ *CwdStack,* const u_signed64 ∗ *FilesetIdHint,* u_signed64 ∗ *FilesetId*)

Retrieve an object's fileset id.

Parameters

in	*ThreadContext*	Thread Context
in	*RequestId*	Request Id
in	*Ucred*	User Credentials
in	*ChaseFlags*	Flags which determine traversal policy
in	*ObjHandle*	Object Handle
in	*Path*	Pathname for lookup
in	*CwdStack*	Cwd Stack
in	*FilesetIdHint*	Fileset id hint
in,out	*FilesetId*	Fileset id

This function handles fileset id processing for trashcans in situations where it cannot be easily discerned from cached data. A fileset id hint may come in from a higher level function, and if it exists it will be used. Otherwise we will get the fileset attribute from the Core Server.

Note

This is a point where additional caching could be useful to preserve delete/rename performance, but we're not at that point yet.

Returns

The filesetid of the requested object

11.27.2.34 uint64_t API_GetOpenTableSlot (filetable_t ∗ *Ftptr*)

Find an open file table spot.

Parameters

in	*Ftptr*	File table

Returns

An open file table slot, or a number larger than the table size if no open slot is available.

Note

We could have a thread or something run and periodically free up unused, allocated slots to save on memory. We could also compact the table if we found ourselves where most of the open files were at the front of the table.

11.27.2.35 char∗ API_HexDecode (const char ∗ *buffer,* int ∗ *len*)

Decode a hex buffer as binary into a buffer string.

Parameters

in	*buffer*	Hex buffer
out	*len*	Length of the binary buffer

Returns

The buffer, decoded into binary, or NULL.

11.27.2.36 char∗ API_HexEncode (const char ∗ *buffer,* int *buflen*)

Encode a digest buffer into a hex string.

Parameters

`in`	*buffer*	Digest buffer
`in`	*buflen*	Length of the digest buffer

Returns

The buffer, encoded as a hex string or NULL

Note

This function will only stop when buflen bytes have been read. The string is assumed to be at least *buflen* bytes worth of data to be encoded into a hex format.

11.27.2.37 int API_InitializeServerIOD (const hpss_IOD_t ∗ *IODPtr,* IOD_t ∗ *RetIODPtr*)

Initiliazes a server IOD.

This routine is called to initialize an HPSS server IOD.

Parameters

`in`	*IODPtr*	client IOD pointer
`in,out`	*RetIODPtr*	newly initialize IOD pointer

Return values

0	Success
ENOMEM	If memory allocation fails

11.27.2.38 void API_InsertInFreeList (hpss_ilist_t ∗ *List,* uint64_t *Des*)

Put an entry back into the free list.

Parameters

`in`	*List*	List to add to
`in`	*Des*	File descriptor to place back in the table

Note

The file table should be locked.

11.27.2.39 int API_InterruptDataTask (open_bf_desc_t ∗ *OpenBFDescPtr*)

Writes data to a piped read descriptor.

Will cause a piped read descriptor to have activity by writing a small amount of data to it.

Parameters

in	*OpenBFDescPtr*	Object Handle pointer

Return values

0	Success.
Other	Write to data thread interrupt failed.

Note

This function should be used with API_CleanupInterruptBuffer so that subsequent operations will behave normally. See the note in API_CleanupInterruptBuffer for more details.

11.27.2.40 int32_t API_LockCondition (api_condition_variable_t * *Lock,* const char * *Function*)

Uses a condition variable to lock access to data.

Lock the data protected by the specified Lock structure.

Parameters

in	*Lock*	Structure for data access control
in	*Function*	Calling function for logging

Return values

0	Success

11.27.2.41 void API_LockMutex (pthread_mutex_t * *MutexPtr*)

Locks a pthread mutex.

This routine locks a pthread mutex. If an error is encountered, a message is written to standard error and exit() is called.

Parameters

in	*MutexPtr*	pointer to mutex

Return values

0	- Success

11.27.2.42 int API_net_bind (int *sockfd,* hpss_sockaddr_t * *addr,* char * *errbuf,* size_t *errbuflen*)

Bind wrapper for busy retry handling.

Parameters

in	*sockfd*	Socket Descriptor
in	*addr*	Bind Address
in,out	*errbuf*	Error Buffer
in	*errbuflen*	Length of error buffer

Note

This function will retry the EADDRINUSE condition using the configured API busy delay and retry values.

11.27.2.43 int API_Net_CreateAddr_INADDR_ANY (hpss_sockaddr_t * *SockAddr,* int *Port,* hpss_reqid_t *ReqId*)

Creates a socket address for listen / bind calls.

Creates a socket address structure suitable for listen / bind calls. Compatible with either IPV4 or IPV6.

Parameters

`in,out`	*SockAddr*	
`in`	*Port*	
`in`	*ReqId*	

Return values

0	Successfully created hpss_sockaddr_t
-EIO	Error getting address info

Note

The hpss net library sets the AI_PASSIVE flag anyway, but we are still going to do it so we don't hide what's going on here. Consult the getaddrinfo man page for the secret formula for generating a INADDR_ANY sockaddr.

11.27.2.44 void API_ObjHandleToString (const ns_ObjHandle_t * *Handle,* char * *Buffer*)

Displays an NS object handle in text form.

The "API_ObjHandleToString" routine will return a textual representation of the specified NS object handle.

Parameters

`in`	*Handle*	Object Handle pointer
`in`	*Buffer*	Buffer to use

11.27.2.45 int API_OpenListenDesc (int *Type,* int * *ListenDesc,* data_addr_t * *ListenAddr*)

Opens a list port for a thread.

Parameters

`in`	*Type*	Type of descriptor to open
`out`	*ListenDesc*	Returned open socket
`out`	*ListenAddr*	Returned socket identification

The API_OpenListenDesc routine is called by a thread when that thread wishes to open a unique listen port. A new descriptor will be opened and then bound to a unique port and then a listen will be put out on the port.

Return values

0	NoError
HPSS_ECOMM	Communication error building the socket
HPSS_EINVAL	Type of descriptor

11.27.2.46 int API_OpenThrdCntlDesc (open_bf_desc_t ∗ *OpenBFDescPtr*)

Creates a thread control pipe for ending a data thread.

Create the thread control pipe to be used to end a data thread.

Parameters

`in`	*OpenBFDescPtr*	Open Bitfile Descriptor

Return values

0	Success
Other	Error returned from pipe() or fcntl()
EBADF	Input thread control files are in an invalid state

11.27.2.47 int API_PerformRetry (retry_cb_t ∗ *RetryCB,* int *Error*)

Retries a call to an API.

The API_PerformRetry routine is called when retrying a call to some interface (API). A caller passes in a retry control block and the error returned from the last failed API. In the control block a retry class is specified to indicate information for the upcoming retry. The values for the retry class are:

- NON_RETRYABLE Never retry (default)
- RETRYABLE Retry 'NumRetries' number of times with no delay.
- DELAY_RETRYABLE Retry a 'NumRetries' number of times with a 'BusyDelay' second delay between each.
- LIMITED_RETRYABLE Retry a 'LimitedRetries' number of times with no delay.
- GK_RETRYABLE Retry forever with a delay specified by the Gatekeeper
- RQ_RETRYABLE Retry with a delay based on the Request Queue

Parameters

`in`	*RetryCB*	retry control block
`in`	*Error*	error return code

Return values

None.	

Note

The API_RETRY_INIT macro can be use to ensure the retry control block is properly initialized on the first call to API_RETRY. At this time any total retry time is established.

Use the retry utilities in the following manner:

```
retry_cb_t  retry_cb;

API_RETRY_INIT(&retry_cb);
API_RETRY(&retry_cb, error)
{
  error = function(....);
  case Retryable_Error:
    retry_cb->retry_class = RETRYABLE;
    break;
  case Limited_Retryable_Error:
    retry_cb->retry_class = LIMITED_RETRYABLE;
    break;
  case Retryable_Delay_Error:
    retry_cb->retry_class = DELAY_RETRYABLE;
    break;
  case GK_Retryable_Error:
    retry_cb->retry_class = GK_RETRYABLE;
    break;
 }
```

If the caller does not specify type retry class in the control block, it will be defaulted to NON_RETRYABLE.

If the 'TotalDelay' configuration variable is non-zero then the total time for any retries is constrained by these values.

The default retries values are defined in api_init.c and are currently set at:

- BusyDelay - 15 seconds
- BusyRetries - 3 times
- LimitedRetries - 1 time
- TotalDelay - 0 seconds (disabled)

11.27.2.48 int32_t API_RemoveCachedTrash (hpss_reqid_t *RequestID*, const sec_cred_t ∗ *Ucred*, const u_signed64 ∗ *FSID*, int *OptionFlags*)

Remove a cached trashcan entry.

Parameters

in	*RequestID*	Client Request ID
in	*Ucred*	User's Credentials
in	*FSID*	Fileset Id
in	*OptionFlags*	The deletion flags

This function removes a cached trashcan entry based upon the type of trashcan used. It should only be called in response to a trashcan cache miss from the Core Server, which is documented as returning an HPSS_EBADTRASH. For other purposes such as cleanup, the cache functions which clear out the trash should be used instead. The caches are self-cleaning and should remove old entries if they become full so there's no need to micromanage cache entries. The function does not do anything to handle errors in cache removal rather than log them at this point.

Return values

HPSS_E_NOERROR	Success
HPSS_EINVAL	The OptionFlags do not specify whether home or fileset trash should be cleared.

11.27.2.49 int API_RetrieveTrashcanHandle (apithrdstate_t * *ThreadContext*, hpss_reqid_t *RequestID*, const sec_cred_t * *Ucred*, const ns_ObjHandle_t * *ParentHandle*, const u_signed64 * *FSId*, api_cwd_stack_t * *CwdStack*, ns_ObjHandle_t ** *TrashcanContainerHandle*, ns_ObjHandle_t ** *TrashcanHandle*, int * *OptionFlags*)

Find the appropriate trashcan to use based upon the fileset, parent handle, and user.

Parameters

`in`	*ThreadContext*	Client API Thread Context
`in`	*RequestID*	Client Request ID
`in`	*Ucred*	User's Credentials
`in`	*ParentHandle*	Parent Object Handle
`in`	*FSId*	Fileset Id
`in`	*CwdStack*	Current Working Directory Stack
`in,out`	*Trashcan-ContainerHandle*	This is the object which contains the ".Trash" directory. For a user homedir situation it is the homedir, for a fileset it is the fileset root.
`in,out`	*TrashcanHandle*	This is the actual trashcan directory.
`in,out`	*OptionFlags*	The deletion flags which will be sent to the Core Server. The type of trash handle needed will be identified for Core Server lookup/creation.

This function finds out where the user's trashcan is at this point and gets an object handle to it so it can be passed with the delete information.

First, check the caches. If the caches are empty then we do the following:

We must determine if we are in the fileset which contains the user's homedir, and if the user has an HPSS homedir. If not, then standard fileset rules apply, otherwise the user's trash in their homedir should be used.

First, the fileset id of the object in question is needed. If the fileset is the same as the current fileset, it can be retrieved from cache, other wise it can be looked up.

1. Identify the trashcan required (homedir vs. fileset) 2. Traverse the trashcan path for the appropriate fileset (homedir or fileset).

The Core Server will require at least the TrashcanContainerHandle be set. If this is not set, then a delete with trashcans enabled will fail. The TrashcanHandle can be NULL - this simply means the trashcan has not been created yet, and the Client API relies upon the Core Server to properly create the trashcan directory structures.

Return values

HPSS_E_NOERROR	Success
HPSS_ENOTDIR	The trashcan location is not a directory
Other	Error from hpss_GetPathHandle

11.27.2.50 int API_SetFileTableBusy (filetable_t * *Ftptr*, int *Fildes*, int *FlagChecks*)

Mark a file table entry as busy.

Marks a file table entry as busy if it passes a set of basic conditions (valid range, entry not already busy) plus a set of checks provided by the caller.

Parameters

in	*Ftptr*	File table
in	*Fildes*	Open file descriptor
in	*FlagChecks*	Flags to check before setting the descriptor busy

Return values

0	Success
-EBADF	Not an open bitfile handle, file descriptor out of range, file open with wrong mode
-EBUSY	Some other thread is manipulating the entry
-ESTALE	Connection is no longer valid
-EFAULT	No file table pointer or a null object handle

Note

This function requires at least one valid flag be provided. The bounds of the file table are always checked, as is the busy state.
The flags used to check the various file table states are described in api_internal.h
The file table mutex should be locked prior to calling this function

11.27.2.51 void API_SetLastHPSSErrno (apithrdstate_t ∗ *Threadcontext,* int32_t *HPSSErrno,* const char ∗ *Func,* hpss_reqid_t *RequestId*)

Sets the current HPSS errno state.

Parameters

in	*Threadcontext*	API Thread Context
in	*HPSSErrno*	HPSS Error Code
in	*Func*	Name of failing function
in	*RequestId*	Request Id of failing request

Sets the current HPSS errno state.

11.27.2.52 int API_TranslateError (int32_t *Error,* retry_cb_t ∗ *RetryCB*)

Translates an HPSS error into a POSIX error.

Parameters

in	*Error*	error value
in	*RetryCB*	retry control block

The API_TranslateError routine is called to translate an HPSS error number into a POSIX compliant error number, and identify the retry class.

Return values

None.	

11.27.2.53 int32_t API_UnlockCondition (api_condition_variable_t ∗ *Lock,* const char ∗ *Function*)

Unlocks data controlled by a condition variable.

Parameters

in	*Lock*	Structure for data access control
in	*Function*	Calling function for logging

Unlock the data protected by the specified Lock structure

Return values

0	Success

11.27.2.54 void API_UnlockFileTableEntries (iod_srcsinkdesc_t ∗ *SrcSinkPtr,* filetable_t ∗ *FileTabPtr,* hpss_ilist_t ∗ *FildesLocked,* int *NumLocked,* int *LockTable*)

Unlocks entries in a client file table if previously marked busy.

The "UnlockFileTableEntries" routine will mark file table entries free which have been marked as busy by the caller.

Parameters

in	*SrcSinkPtr*	start of src/sink descriptor list
in	*FileTabPtr*	pointer to current file table
in	*FildesLocked*	indicates files to be unlocked
in	*NumLocked*	number of files to unlock
in	*LockTable*	should we lock/unlock the table?

11.27.2.55 void API_UnlockMutex (pthread_mutex_t ∗ *MutexPtr*)

Unlocks a pthread mutex.

This routine unlocks a pthread mutex. If an error is encountered, a message is written to standard error and exit() is called.

Parameters

in	*MutexPtr*	pointer to mutex

Return values

0	- Success

11.27.2.56 hpss_read_queue_list_t hpss_GetIntersectedList (hpss_read_queue_list_t ∗ *List,* hpss_read_queue_list_t ∗ *Partial*)

Return a list whose content is all items in the list which do appear in the sublist.

Parameters

in	*List*	List of all items to process
in	*Partial*	Sublist of items to include

Returns

A new allocation of list items

Note

The new list array must be freed

11.27.2.57 hpss_read_queue_list_t hpss_RemoveIntersectionFromList (hpss_read_queue_list_t ∗ *List*, hpss_read_queue_list_t ∗ *Partial*)

Return a list whose content is all items in the list which do not appear in the sublist.

Parameters

in	*List*	List of all items to process
in	*Partial*	Sublist of items to remove

Returns

A new allocation of list items

Note

The new list array must be freed

11.28 api_misc_admin.c File Reference

Functions for interfacing with the Core Server's misc. functions.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_MiscAdmin (uint32_t Function, char ∗TextP, ns_AdminConfArray_t ∗ValueConfArrayP)
  Accesses the Core Server's miscellaneous functions.

11.28.1 Detailed Description

Functions for interfacing with the Core Server's misc. functions.

11.29 api_mkdir.c File Reference

Functions for creating directories.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
#include "acct_av_lib.h"
```

Functions

- int hpss_Mkdir (const char ∗Path, mode_t Mode)

    Creates a new directory.
- int hpss_MkdirHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, mode_t Mode, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗HandleOut, hpss_vattr_t ∗AttrOut)

    Creates a new directory.

11.29.1 Detailed Description

Functions for creating directories.

11.30 api_open.c File Reference

Functions for opening HPSS files.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <netdb.h>
#include <netinet/in.h>
#include "hpss_soid_func.h"
#include "hpss_pthread.h"
#include "pdata.h"
#include "api_internal.h"
#include "hpss_api.h"
#include "acct_av_lib.h"
#include "hpss_Getenv.h"
```

Functions

- int hpss_Creat (const char ∗Path, mode_t Mode, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

    Create a file and open it.
- int hpss_Create (const char ∗Path, mode_t Mode, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

    Create an HPSS file.
- int hpss_CreateHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, mode_t Mode, sec_cred_t ∗Ucred, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut, hpss_vattr_t ∗AttrsOut)

    Create a file using a handle.
- int hpss_Open (const char ∗Path, int Oflag, mode_t Mode, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

    Optionally create and open an HPSS file.

- int hpss_OpenBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, int Oflag, const sec_cred_t ∗Ucred)

  Open an HPSS file by bitfile id.
- int hpss_OpenBitfileVAttrs (const hpss_vattr_t ∗FileAttrs, int Oflag, const sec_cred_t ∗Ucred, hpss_cos_-hints_t ∗HintsOut, uint64_t ∗SegmentSize)

  Open a bitfile using vattrs.
- int hpss_OpenHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, int Oflag, mode_t Mode, sec-_cred_t ∗Ucred, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut, hpss_vattr_t ∗AttrsOut)

  Optionally create and open an HPSS file relative to a given directory.
- int hpss_ReopenBitfile (int Fildes, const bfs_bitfile_obj_handle_t ∗BitfileObj, int Oflag, const sec_cred_t ∗Ucred)

  Open an HPSS file by bitfile object handle.

11.30.1 Detailed Description

Functions for opening HPSS files.

11.31 api_opendir.c File Reference

Functions for opening directories.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "hpss_uuid.h"
#include "api_internal.h"
```

Functions

- int hpss_Opendir (const char ∗DirName)

  Opens a directory stream.
- int hpss_OpendirHandle (const ns_ObjHandle_t ∗DirHandle, const sec_cred_t ∗Ucred)

  Open a directory via a handle.

11.31.1 Detailed Description

Functions for opening directories.

11.32 api_path.c File Reference

Functions for traversing paths.

```
#include <unistd.h>
#include <inttypes.h>
#include "hpss_api.h"
#include "hpss_ObjHandle.h"
#include "api_internal.h"
```

Functions

- void API_FreeCwdStack (api_cwd_stack_t ∗Stack)

  Frees the memory allocated for a cwd stack.
- int API_InitCwdStack (api_cwd_stack_t ∗Stack, uint32_t Local)

  Initializes a cwd stack.
- int API_PathFromCwdStack (api_cwd_stack_t ∗Stack, uint32_t Size, char ∗Path)

  Builds a path from the cwd stack.
- int API_SetCwdStack (api_cwd_stack_t ∗CwdStack, api_cwd_stack_t ∗FromStack)

  Copies the contents of a cwd stack.
- int API_TraversePath (apithrdstate_t ∗ThreadContext, hpss_reqid_t RequestID, const sec_cred_t ∗Ucred, const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, api_cwd_stack_t ∗CwdStack, uint32_t TraversalFlags, uint32_t XAttrFlags, uint32_t XAttrStoreLevel, hpss_AttrBits_t SelectFlags, hpss_AttrBits_t ParentSelectFlags, api_cwd_stack_t ∗RetCwdStack, ns_ObjHandle_t ∗RetHandle, hpss_Attrs_t ∗RetAttrs, ns_ObjHandle_t ∗RetParentHandle, hpss_Attrs_t ∗RetParentAttrs, char ∗RetObjectName, bf_sc_attrib_t ∗XAttr)

  Locates an object in HPSS namespace.
- int hpss_GetFullPath (const ns_ObjHandle_t ∗ObjHandle, char ∗∗FullPath)

  Provides the full path back to an object from the root of roots.
- int hpss_GetFullPathBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, char ∗∗FullPath)

  Provides a full path back to a bitfile from the root of roots.
- int hpss_GetFullPathBitfileLen (const bfs_bitfile_obj_handle_t ∗BitfileObj, char ∗Path, size_t BufLen)

  Provides a full path back to a bitfile from the root of roots.
- int hpss_GetFullPathHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, char ∗∗FullPath)

  Provides the full path back to an object from the root of roots.
- int hpss_GetFullPathHandleLen (const ns_ObjHandle_t ∗ObjHandle, const char ∗ObjPath, char ∗Path, size_t BufLen)

  Provides the full path back to an object from the root of roots.
- int hpss_GetFullPathLen (const ns_ObjHandle_t ∗ObjHandle, char ∗Path, size_t BufLen)

  Provides the full path back to an object from the root of roots.
- int hpss_GetPathHandle (const ns_ObjHandle_t ∗ObjHandle, ns_ObjHandle_t ∗FilesetRootHandle, char ∗Path)

  Retrieves the pathname and root fileset that correspond with an Objhandle.
- int hpss_GetPathHandleObjHash (const object_id_hash_t ∗ObjIdHash, uint32_t SubsysId, ns_ObjHandle_t ∗FilesetRootHandle, ns_ObjHandle_t ∗ObjHandle, char ∗Path)

  Retrieves fileset, object handle, and path information using an object and subsystem.
- int hpss_GetPathObjHash (const object_id_hash_t ∗ObjIdHash, uint32_t SubsysId, ns_ObjHandle_t ∗FilesetRootHandle, char ∗Path)

  Retrieves the pathname and root fileset that correspond to an Object and Subsystem.
- char ∗ NormalizePath (char ∗∗Path)

  Normalizes all '.'s and '..'s of a path.

11.32.1 Detailed Description

Functions for traversing paths.

11.32.2 Function Documentation

11.32.2.1 void API_FreeCwdStack (api_cwd_stack_t ∗ *Stack*)

Frees the memory allocated for a cwd stack.

This function is called to free memory allocated for the specified cwd stack.

Parameters

`in`	*Stack*	Pointer to stack

Return values

None.	

11.32.2.2 int API_InitCwdStack (api_cwd_stack_t ∗ *Stack*, uint32_t *Local*)

Initializes a cwd stack.

Parameters

`in`	*Stack*	Pointer to cwd stack
`in`	*Local*	Thread local stack flag

This function is called to initialize a cwd stack by setting its size to 0 and the top pointer to NULL.

Return values

None.	

11.32.2.3 int API_PathFromCwdStack (api_cwd_stack_t ∗ *Stack*, uint32_t *Size*, char ∗ *Path*)

Builds a path from the cwd stack.

This function is called to build a path from all the path fragments on the inputted stack.

Parameters

`in`	*Stack*	Pointer to cwd stack
`in`	*Size*	Size of memory for path
`out`	*Path*	Returned path

Return values

-ENOMEM	if can't allocate memory for stack entry
0	otherwise

Note

> This routine is called only with global stacks, so we don't worry about checking for the local flags before locking and unlocking the stack's mutex.

11.32.2.4 int API_SetCwdStack (api_cwd_stack_t ∗ *CwdStack*, api_cwd_stack_t ∗ *FromStack*)

Copies the contents of a cwd stack.

This function is used to copy the contents of a current working directory stack.

Parameters

in	*CwdStack*	Pointer to cwd stack
in	*FromStack*	Pointer to from stack

Return values

ENOMEM	If can't allocate memory for stack entry
0	otherwise

11.32.2.5 int API_TraversePath (apithrdstate_t ∗ *ThreadContext*, hpss_reqid_t *RequestID*, const sec_cred_t ∗ *Ucred*, const ns_ObjHandle_t ∗ *ObjHandle*, const char ∗ *Path*, api_cwd_stack_t ∗ *CwdStack*, uint32_t *TraversalFlags*, uint32_t *XAttrFlags*, uint32_t *XAttrStoreLevel*, hpss_AttrBits_t *SelectFlags*, hpss_AttrBits_t *ParentSelectFlags*, api_cwd_stack_t ∗ *RetCwdStack*, ns_ObjHandle_t ∗ *RetHandle*, hpss_Attrs_t ∗ *RetAttrs*, ns_ObjHandle_t ∗ *RetParentHandle*, hpss_Attrs_t ∗ *RetParentAttrs*, char ∗ *RetObjectName*, bf_sc_attrib_t ∗ *XAttr*)

Locates an object in HPSS namespace.

'API_TraversePath' is called to locate a specified object within the HPSS name space. This may require that one or more Core Servers be queried while traversing the requested path. If the object is located, any requested information (handle, attributes or authorization) is returned. If the 'RetCwdStack' pointer is not NULL and the object is found to be a directory with search permissions, then the list of junctions and symlinks crossed to get to the object is saved in the current thread context (ThreadContext->CwdStackPtr).

Parameters

in	*ThreadContext*	current context
in	*RequestID*	request id
in	*Ucred*	user credentials
in	*ObjHandle*	object handle
in	*Path*	pathname
in	*CwdStack*	cwd stack
in	*TraversalFlags*	path traversal flags
in	*XAttrFlags*	extended attrs flags
in	*XAttrStoreLevel*	extended attrs args
in	*SelectFlags*	object attributes bits
in	*ParentSelect-Flags*	parent attribute bits
out	*RetCwdStack*	returned cwd stack
out	*RetHandle*	object handle
out	*RetAttrs*	object attributes
out	*RetParent-Handle*	parent handle
out	*RetParentAttrs*	parent attributes
out	*RetObjectName*	object name
out	*XAttr*	extended attrs

Return values

0	Success
Other	Error

Note

If the 'RetParentAttrs' input parameter is non-null, then the parent attributes for the object will be returned. If the object has no parent (a file set root), then the attributes of the object itself will be returned in this parameter.

If the 'RetObjectName' input parameter is non-null then the object name will be returned. This input must be a string at least HPSS_MAX_FILE_NAME in length.

This routine works by creating a stack of junctions and symlinks crossed to get to the specified location. Each stack element contains a handle and a path. The handle points at the base directory from which the path starts. Each time a junction or symlink is crossed, a new stack element is pushed on the stack. Each time the path at the top of the stack is backed over, a element is pop off the stack. This pushing and popping continues until a requested component can't be located, the path is successfully traversed, or until the stack grown beyound the established limit.

Since the advent of filesets and junctions the client library can no longer depend on the NS to supply it with a meaningful path that corresponds to a current working directory handle. Therefore, the library must use the afore mentioned directory stack to build a current working directory path on request.

```
Algorithm:
     If ( no path specified )
        Get the attributes for the supplied handle;
     Else
     {
        Assign 'new_path' and 'new_handle' using 'Path' & 'ObjHandle';
        While ( haven't crossed too many junctions (or symlinks) )
        {
           If ( current working directory stack is not empty )
              Set the don't backup flags;
           If ( this is a change directory )
              Set the break traversal on symlinks flags;

           Get the attributes for 'new_path' & 'new_handle'

           If ( no error )
           {
              If ( the object is a junction & chasing junctions )
                 Append '.' to 'new_path' and continue;
              If ( this is a change directory )
              {
                 Pop the top element off the stack, if there is one;
                 Append the 'new_path' to the top stack path;
                 Push the top element back off the stack;
                 Normalize all the '.' & '..' out of the stack;
                 # For change directories, we ask that the NS
                 # not chase symlinks that are the last element
                 # of a path. We want to do this ourselves in order
                 # to maintain a current working directory that
                 # that look likes what the user would expect.
                 # Because of this we need to chase the symlinks
                 # ourselves and use the returned handle and
                 # attributes, but keep the symlink name in the
                 # current directory stack.
                 If ( the object is a symlink )
                 {
                    Copy the current work directory stack;
                    Read the contents of the symlink;
                    Pop the top element off the stack;
                    Remove the symlink component from the stack path;
                    Replace it with the symlink contents;
                    Call API_TraversePath() with the new stack;
                 }
                 Copy local copy of current working directory stack
                    to thread state current working directory stack
              }
              Break from loop;
           }
           Else if ( there is more path to traverse )
           {
              Here we have a remaining handle and path returned
                 from the NS.
              # Determine how to handle the remaining path by
              # comparing each component of the remaining path
              # with the path used in the get attributes call
              # (new path).
              Separate the common component of the remaining and new path;
              If ( a symlink was crossed )
              {
                 Create a base path that contains the part of the remaining
```

```
                path that is not contained in the new path;
             Call this routine recursively to get the handle for the
                base path (symbolic link);
             Set the remaining handle from the returned handle;
          }
          Pop the top element off the current directory stack;
          Append the new path (portion not in the remaining path) to
             the top element path.
          Push this element back on the stack.
          Push an element back on the stack that contains the
             the remaining handle and an empty path;
          The new path now becomes the component of the remaining
             path that were contained in the new path.
       }
       Else if ( backed over the input path )
       {
          Pop the top element off the top of the stack;
          If ( the top element has an empty path )
          {
             Pop the next element off the top of the stack;
          }
          Push the handle from the last element pop, back on
             the stack with an empty path;
          Remove the last component from the first non-empty
             path that was at the top of the stack.
          Assign this path appended with any remaining path
             returned to the new path;
       }
       Else
       {
          Break on any other errors;
       }
       If ( the stack size is too large )
          Set error to EMLINK and break from loop;
    }
    Free any allocates resources;
 }
Return requested attributes;
```

11.32.2.6 char∗ NormalizePath (char ∗∗ *Path*)

Normalizes all '.'s and '..'s of a path.

This function is called to normalize all DOTs and DOTDOTs out of the specified path. If the path jumps back over the starting point the function will return a NULL and the remainder of the path.

Parameters

in	*Path*	Pointer to path to normalize

Return values

None.	

11.33 api_pio.c File Reference

Functions for performing Parallel IO.

```
#include <inttypes.h>
#include <stdlib.h>
#include <unistd.h>
#include <stddef.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include "hpss_errno.h"
#include "hpss_pthread.h"
#include "hpss_thread_safe.h"
#include "hpss_interop.h"
#include "hpss_uuid.h"
#include "hpss_iod_clnt.h"
#include "traniod.h"
#include "pdata.h"
#include "hpss_Getenv.h"
#include "san3p.h"
#include "san3p_util.h"
#include "api_internal.h"
#include "hpss_memalign.h"
```

Functions

- int hpss_PIOEnd (hpss_pio_grp_t StripeGroup)

  End and clean up a parallel IO group context.
- int hpss_PIOExecute (int Fd, uint64_t FileOffset, uint64_t Size, hpss_pio_grp_t StripeGroup, hpss_pio_gapinfo_t ∗GapInfo, uint64_t ∗BytesMoved)

  Begin a PIO data transfer.
- int hpss_PIOExecuteItem (int Fd, uint64_t FileOffset, uint64_t Size, const hpss_pio_grp_t StripeGroup, hpss_pio_gapinfo_t ∗GapInfo, uint64_t ∗BytesMoved, hpss_read_queue_item_t ∗Item)

  Begin a PIO data transfer, with a reference to a running read queue item.
- int hpss_PIOExportGrp (const hpss_pio_grp_t StripeGroup, void ∗∗Buffer, unsigned int ∗BufLength)

  Export a parallel IO group context to a buffer.
- int hpss_PIOFinalize (hpss_pio_grp_t ∗StripeGroup)

  Safely end and clean up a parallel IO group context.
- int hpss_PIOGetPartError (hpss_pio_grp_t StripeGroup, int ∗Error)

  Retrieves a participant error code.
- int hpss_PIOGetRequestID (hpss_pio_grp_t StripeGroup, hpss_reqid_t ∗RequestID)

  Retrieves the request ID for a group.
- int hpss_PIOImportGrp (const void ∗Buffer, unsigned int BufLength, hpss_pio_grp_t ∗StripeGroup)

  Import a group which was exported with hpss_PIOExportGrp.
- int hpss_PIORegister (uint32_t StripeElement, const hpss_sockaddr_t ∗DataNetSockAddr, void ∗DataBuffer, uint32_t DataBufLen, hpss_pio_grp_t StripeGroup, const hpss_pio_cb_t IOCallback, const void ∗IOCallbackArg)

  Register a PIO stripe participant.
- int hpss_PIOStart (hpss_pio_params_t ∗InputParams, hpss_pio_grp_t ∗StripeGroup)

  Start a new parallel I/O group context.
- void PIOInit (void)

  PIOInit.

11.33.1 Detailed Description

Functions for performing Parallel IO.

11.33.2 Function Documentation

11.33.2.1 void PIOInit (void)

PIOInit.

This function is called to do initialization required by an I/O participant.

11.34 api_posix_lseek.c File Reference

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "u_signed64.h"
#include "hpss_api.h"
#include "hpss_posix_api.h"
#include "api_internal.h"
```

Functions

- long hpss_PosixLseek (int Fildes, long Offset, int Whence)

  Set the file offset for an open file handl.

11.35 api_posix_open.c File Reference

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <netdb.h>
#include <netinet/in.h>
#include "hpss_soid_func.h"
#include "u_signed64.h"
#include "hpss_Getenv.h"
#include "hpss_pthread.h"
#include "pdata.h"
#include "hpss_api.h"
#include "hpss_posix_api.h"
#include "api_internal.h"
#include "acct_av_lib.h"
```

Functions

- int hpss_CreateWithHints (char ∗Path, mode_t Mode, hpss_cos_hints_t ∗HintsIn, hpss_cos_priorities_-t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

  Create an HPSS file with hints.
- int hpss_OpenWithHints (char ∗Path, int Oflag, mode_t Mode, hpss_cos_hints_t ∗HintsIn, hpss_cos_-priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

  Open a file using HPSS hints.
- int hpss_PosixCreate (char ∗Path, mode_t Mode)

  Create an HPSS file.
- int hpss_PosixOpen (char ∗Path, int Oflag, mode_t Mode)

  Open an HPSS file.

11.36 api_purge.c File Reference

Functions for purging files.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Migrate (int Fildes, uint32_t SrcLevel, uint32_t Flags, uint64_t ∗RetBytesMigrated)

  Migrates data in an open file from a specified hierarchy level.
- int hpss_Purge (int Fildes, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, uint32_t Flags, uint64_t ∗RetBytesPurged)

  Purges data in an open file from a specified hierarchy level.

11.36.1 Detailed Description

Functions for purging files.

11.37 api_purgelock.c File Reference

Functions for setting the purgelock flag on files.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_PurgeLock (int Fildes, purgelock_flag_t Flag)

  Locks or unlocks a file's purge state.

11.37.1 Detailed Description

Functions for setting the purgelock flag on files.

11.38 api_purgeonmigrate.c File Reference

Functions for setting the purgeonmigrate flag on files.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_PurgeOnMigrate (int Fildes, purgeonmigrate_flag_t Flag)

  Sets or clears the purge on migrate flag.

11.38.1 Detailed Description

Functions for setting the purgeonmigrate flag on files.

11.39 api_rddir.c File Reference

Functions for reading directory entries.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include "hpss_dirent.h"
#include "hpss_api.h"
#include "hpss_uuid.h"
#include "api_internal.h"
#include "ns_interface_defs.h"
```

Functions

- int hpss_Getdents (const ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_t ∗Ucred, uint32_t BufferSize, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

  Retrieve directory entries with reliable entry offsets.

- int hpss_GetdentsAttrs (ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_t ∗Ucred, uint32_t BufferSize, uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entries and object attributes in an offset-reliable manner.
- int hpss_ReadAttrs (const int Dirdes, const uint64_t OffsetIn, const uint32_t BufferSize, const uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entries and object attributes from an open directory stream.
- int hpss_ReadAttrsHandle (const ns_ObjHandle_t ∗ObjHandle, const uint64_t OffsetIn, const sec_cred_t ∗Ucred, const uint32_t BufferSize, const uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entry attributes using a handle.
- int hpss_ReadAttrsPlus (int Dirdes, uint64_t OffsetIn, uint32_t BufferSize, hpss_readdir_flags_t Flags, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entries and object attributes from an open directory stream.
- int hpss_Readdir (int Dirdes, hpss_dirent_t ∗DirentPtr)

  Read directory entries from an open directory stream.
- int hpss_ReaddirHandle (const ns_ObjHandle_t ∗ObjHandle, const uint64_t OffsetIn, const sec_cred_t ∗Ucred, const uint32_t BufferSize, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

  Read directory entries using a handle.
- int hpss_ReaddirPlus (int Dirdes, uint64_t OffsetIn, uint32_t BufferSize, hpss_readdir_flags_t Flags, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

  Read directory entries using a handle.
- int hpss_ReadRawAttrsHandle (const ns_ObjHandle_t ∗ObjHandle, const uint64_t OffsetIn, const sec_cred_t ∗Ucred, const uint32_t BufferSize, const uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read raw attributes using a handle; does not cross junctions.
- int hpss_ReadTrash (signed32 SubsystemId, const unsigned32 ∗UserIdP, const unsigned32 ∗RealmIdP, unsigned32 BufferSize, unsigned32 ∗End, ns_TrashEntry_t ∗TrashPtr)

  Read trash entries for a subsystem.
- int hpss_Rumble (signed32 SubsystemId, timespec_t ∗TimeP, unsigned32 BufferSize, unsigned32 ∗End, ns_RumbleEntry_t ∗RumblePtr)

  Read rumble entries for a subsystem.

11.39.1 Detailed Description

Functions for reading directory entries.

11.39.2 Function Documentation

11.39.2.1 int hpss_Rumble (**signed32** *SubsystemId,* **timespec_t** ∗ *TimeP,* **unsigned32** *BufferSize,* **unsigned32** ∗ *End,* **ns_RumbleEntry_t** ∗ *RumblePtr*)

Read rumble entries for a subsystem.

The 'hpss_Rumble' function returns an array of rumble entries, RumblePtr, representing changes to the HPSS namespace since TimeP for subsystem SubsystemId. The number of entries is limited by BufferSize, which describes the size of the allocated RumblePtr structure.

Parameters

in	*SubsystemId*	the subsystem for the core server
in	*TimeP*	return changes after time specified
in	*BufferSize*	size of output buffer (in bytes)
out	*End*	hit end of rumble entries
out	*RumblePtr*	rumble entry information

Return values

>=0	No error. Return value is the number of entries copied to the output buffer.
-ERANGE	The buffer size was not large enough to return entries
-EFAULT	One of the End or RumblePtr parameters is NULL.
-EINVAL	BufferSize is zero.

Note

rumble paths are from their fileset root. For systems which make use of filesets this may not be an absolute path. To obtain an absolute path, the handle should be traversed back to the root of roots. See hpss_GetFull-Path() for more information.

11.40 api_rdlink.c File Reference

Functions for reading links.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Readlink (const char ∗Path, char ∗Contents, size_t BufferSize)

  Retrieves the contents of a symbolic link.
- int hpss_ReadlinkHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, char ∗Contents, size_t BufferSize, const sec_cred_t ∗Ucred)

  Retrieves the contents of a symbolic link.

11.40.1 Detailed Description

Functions for reading links.

11.40.2 Function Documentation

11.40.2.1 int hpss_Readlink (const char ∗ *Path,* char ∗ *Contents,* size_t *BufferSize*)

Retrieves the contents of a symbolic link.

The 'hpss_Readlink' function returns the contents of the named symbolic link.

Parameters

in	*Path*	Name of the link
out	*Contents*	Contents of the link
in	*BufferSize*	Size, in bytes, of Contents

Return values

0	Contents contains symlink data.
-EACCES	Search permission is denied on a component of the path prefix, or read permission is denied on the symbolic link.
-EFAULT	The Path or Contents parameter is a NULL pointer.
-EINVAL	The specified file is not a symbolic link.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The specified path name does not exist.
-ENOTDIR	A component of the Path prefix is not a directory.
-ERANGE	The size of the Contents buffer is not big enough to contain the contents of the symbolic link or the value of the BufferSize parameter is zero.

11.40.2.2 int hpss_ReadlinkHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* char ∗ *Contents,* size_t *BufferSize,* const sec_cred_t ∗ *Ucred*)

Retrieves the contents of a symbolic link.

The 'hpss_ReadlinkHandle' function returns the contents of the symbolic link with the name 'Path' (taken relative to 'ObjHandle').

Parameters

`in`	*ObjHandle*	handle of parent
`in`	*Path*	Name of symlink
`in`	*BufferSize*	Size, in bytes, of Contents
`out`	*Contents*	Contents of the link
`in`	*Ucred*	user credentials

Return values

0	'Contents' contains the symlink data.
-EACCES	Search permission is denied on a component of the path prefix, or read permission is denied on the symbolic link.
-EFAULT	The ObjHandle, Path or Contents parameter is a NULL pointer.
-EINVAL	The specified file is not a symbolic link.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The specified path name does not exist.
-ENOTDIR	A component of the Path prefix is not a directory.
-ERANGE	The size of the Contents buffer is not big enough to contain the contents of the symbolic link or the value of the BufferSize parameter is zero.

11.41 api_rdlist.c File Reference

Functions for directly accessing the HPSS readlist feature.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <unistd.h>
#include <fcntl.h>
#include "hpss_pthread.h"
#include "hpss_api.h"
#include "api_internal.h"
#include "pdata.h"
#include "mvr_protocol.h"
#include "hpss_iod.h"
#include "hpss_interop.h"
#include "san3p.h"
#include "san3p_util.h"
```

Functions

- int hpss_ReadList (const hpss_IOD_t ∗IODPtr, uint32_t Flags, hpss_IOR_t ∗IORPtr)

  Send a read IOD to HPSS.

11.41.1 Detailed Description

Functions for directly accessing the HPSS readlist feature.

11.41.2 Function Documentation

11.41.2.1 int hpss_ReadList (const hpss_IOD_t ∗ *IODPtr*, uint32_t *Flags*, hpss_IOR_t ∗ *IORPtr*)

Send a read IOD to HPSS.

Parameters

in	*IODPtr*	IO description
in	*Flags*	IO specific flags
out	*IORPtr*	results of the IO

The ’hpss_ReadList’ function reads the file data specified by the source descriptor list in the IOD pointed to by ’IODPtr’ and moves the data to destinations specified by the sink descriptor list in the IOD. Results of the request will be returned in the structure pointed to by ’IORPtr’.

Return values

0	- No error.

11.42 api_read.c File Reference

Functions for reading HPSS files.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <fcntl.h>
#include <unistd.h>
#include "hpss_pthread.h"
#include "hpss_api.h"
#include "hpss_memalign.h"
#include "api_internal.h"
#include "pdata.h"
#include "mvr_protocol.h"
#include "traniod.h"
#include "san3p.h"
#include "san3p_util.h"
#include "hpss_interop.h"
```

Functions

- signed32 hpss_PreferredReadOrder (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, hpss_read_queue_list_t ∗ItemsToCheck, int Flags, int Delay, hpss_read_queue_list_t ∗ItemsReady, hpss_read_queue_list_t ∗ItemsError)

    Request to be notified about ready items in a list of reads.
- ssize_t hpss_Read (int Fildes, void ∗Buf, size_t Nbyte)

    Reads a number of bytes from a file.
- ssize_t hpss_ReadItem (int Fildes, void ∗Buf, size_t Nbyte, hpss_read_queue_item_t ∗Item)

    Reads a number of bytes from a file using a read queue item.
- signed32 hpss_ReadQueueAdd (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, hpss_read_queue_item_t ∗Item)

    Add an item from the provided read queue.
- signed32 hpss_ReadQueueAddMany (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, hpss_read_queue_list_t ∗ItemList)

    Add items from the provided read queue.
- signed32 hpss_ReadQueueCreateContext (uint32_t SubsysId, sec_cred_t ∗UserCredsP, hpss_uuid_t ∗ContextP)

    Create a new read queue context.
- signed32 hpss_ReadQueueList (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, hpssoid_t ∗VVID, unsigned32 Start, unsigned32 Max, hpss_read_queue_list_t ∗List)

    List items in the provided read queue.
- signed32 hpss_ReadQueueRemove (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, hpss_reqid_list_t ∗RequestList)

    Remove items from the provided read queue.
- signed32 hpss_ReadQueueRemoveContext (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗UserCredsP, int Flags)

    Remove a read queue context.

11.42.1 Detailed Description

Functions for reading HPSS files.

11.43 api_register.c File Reference

Functions for manipulating bit registers.

```
#include <sys/types.h>
#include <stdarg.h>
#include "hpss_api.h"
```

Functions

- uint64_t API_AddAllRegisterValues (int LastPosition)

    Set all register bit positions up to the position provided.
- uint64_t API_AddRegisterValues (uint64_t InitialValue,...)

    Add values to a bit register.
- uint64_t API_RemoveRegisterValues (uint64_t InitialValue,...)

    Remove values from a bit register.

11.43.1 Detailed Description

Functions for manipulating bit registers.

11.44 api_rename.c File Reference

Functions for renaming files.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Rename (const char ∗Old, const char ∗New)

    Rename an HPSS file or directory.
- int hpss_RenameHandle (const ns_ObjHandle_t ∗OldHandle, const char ∗OldPath, const ns_ObjHandle_t ∗NewHandle, const char ∗NewPath, const sec_cred_t ∗Ucred)

    Rename an HPSS file or directory using an object handle.

11.44.1 Detailed Description

Functions for renaming files.

11.45 api_rewdir.c File Reference

Functions for rewinding an open directory.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Rewinddir (int Dirdes)

    Rewind a directory stream back to the beginning.

11.45.1 Detailed Description

Functions for rewinding an open directory.

11.46 api_rmdir.c File Reference

Functions for removing directories.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int Common_hpss_Rmdir (const char ∗Path, int Immediate)

    Remove a directory, optionally bypassing the trashcan.
- int Common_hpss_RmdirHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, int Immediate)

    Remove a directory using a handle, optionally bypassing the trashcan.
- int hpss_Rmdir (const char ∗Path)

    Remove a directory.
- int hpss_RmdirHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

    Remove a directory using a handle.
- int hpss_RmdirHandleImmediate (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

    Remove a directory using a handle, bypassing the trashcan.
- int hpss_RmdirImmediate (const char ∗Path)

    Remove a directory, bypassing the trashcan.

11.46.1 Detailed Description

Functions for removing directories.

11.46.2 Function Documentation

11.46.2.1 int Common_hpss_Rmdir (const char ∗ *Path,* int *Immediate*)

Remove a directory, optionally bypassing the trashcan.

This function removes the directory named by 'Path'. The directory will only be removed if the directory is empty. It allows trashcan by be optionally bypassed.

Parameters

`in`	*Path*	path of directory
`in`	*Immediate*	Bypass the trashcan

Return values

0	Success
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.
-EBUSY	The named directory is currently in use and cannot be removed.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-ENOTEMPTY	The named directory contains entries other than dot and dot-dot.

11.46.2.2 int Common_hpss_RmdirHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* const sec_cred_t ∗ *Ucred,* int *Immediate*)

Remove a directory using a handle, optionally bypassing the trashcan.

This function removes the directory named by 'Path', Taken relative to the directory indicated by 'ObjHandle'. The directory will only be removed if the directory is empty.

Parameters

`in`	*ObjHandle*	parent object handle
`in`	*Path*	path of directory
`in`	*Ucred*	user credential

Return values

0	Success
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.
-EBUSY	The named directory is currently in use and cannot be removed.
-EFAULT	The Path parameter is a NULL pointer.
-EINVAL	ObjHandle is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.

-ENOTEMPTY	The named directory contains entries other than dot and dot-dot.

11.47 api_stage.c File Reference

Functions for staging data in HPSS.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stddef.h>
#include "hpss_api.h"
#include "api_internal.h"
#include "hpss_soid_func.h"
#include "hpss_uuid.h"
```

Functions

- int API_CopyBatchByStatus (hpss_stage_batch_t ∗Batch, hpss_stage_batch_status_t ∗Status, hpss_stage_batch_t ∗RetryBatch)

  Copy incomplete batch entries into a new retry batch.
- int API_GetBatchStatus (hpss_stage_batch_status_t ∗Status, int ∗Rc)

  Retrieve Batch Status.
- void API_StageBatchFree (hpss_stage_batch_t ∗Batch)

  Free a Batch Stage Request Structure.
- int API_StageBatchGetEntry (hpss_stage_batch_t ∗Batch, int Idx, hpss_stage_t ∗Entry)

  Retrieve Stage Batch Entry.
- int API_StageBatchInit (hpss_stage_batch_t ∗Batch, int len)

  Initialize a Batch Stage Request Structure.
- int API_StageBatchInsertBFHandle (hpss_stage_batch_t ∗Batch, int Idx, hpss_object_handle_t ∗ObjHandle, int FromStorageLevel, int ToStorageLevel, u_signed64 Offset, u_signed64 Length, uint32_t Flags)

  Set a Stage Batch Entry using a Bitfile Handle.
- int API_StageBatchInsertBFObj (hpss_stage_batch_t ∗Batch, int Idx, const bfs_bitfile_obj_handle_t ∗BFObj, int FromStorageLevel, int ToStorageLevel, u_signed64 Offset, u_signed64 Length, uint32_t Flags)

  Set a Stage Batch Entry using a Bitfile ID.
- int API_StageBatchInsertFd (hpss_stage_batch_t ∗Batch, int Idx, int Fd, int FromStorageLevel, int ToStorageLevel, u_signed64 Offset, u_signed64 Length, uint32_t Flags)

  Set a Stage Batch Entry using a File Descriptor.
- void API_StageStatusFree (hpss_stage_batch_status_t ∗Status)

  Free a Batch Stage Status Structure.
- int API_UpdateBatchStatus (hpss_stage_batch_t ∗LocalBatch, hpss_stage_batch_t ∗FinalBatch, hpss_stage_batch_status_t ∗LocalStatus, hpss_stage_batch_status_t ∗FinalStatus)

  Update the batch after processing.
- int hpss_GetAsyncStatus (hpss_reqid_t CallBackId, bfs_bitfile_obj_handle_t ∗BitfileObj, int32_t ∗Status)

  Get the status of a background stage.
- int hpss_GetBatchAsynchStatus (hpss_reqid_t CallBackId, hpss_stage_bitfile_list_t ∗BFObjs, hpss_stage_status_type_t Type, hpss_stage_batch_status_t ∗Status)

  Check the status of files in a batch request.
- int hpss_Stage (int Fildes, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, uint32_t Flags)

  Stage an open HPSS file.

- int hpss_StageBatch (hpss_stage_batch_t ∗Batch, hpss_stage_batch_status_t ∗Status)

  Stage a batch of bitfiles.
- int hpss_StageBatchCallBack (hpss_stage_batch_t ∗Batch, bfs_callback_addr_t ∗CallBackPtr, hpss_reqid_t ∗ReqID, hpss_stage_bitfile_list_t ∗BFObjsList, hpss_stage_batch_status_t ∗Status)

  Stage a batch of bitfiles asynchronously.
- int hpss_StageCallBack (const char ∗Path, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, bfs_callback_addr_t ∗CallBackPtr, uint32_t Flags, hpss_reqid_t ∗ReqID, bfs_bitfile_obj_handle_t ∗BitfileObj)

  Stage an HPSS file in the background.
- int hpss_StageCallBackBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, bfs_callback_addr_t ∗CallBackPtr, uint32_t Flags, hpss_reqid_t ∗ReqID)

  Stage a file in the background using its bitfile id.

11.47.1 Detailed Description

Functions for staging data in HPSS.

11.47.2 Function Documentation

11.47.2.1 int API_CopyBatchByStatus (**hpss_stage_batch_t** ∗ ***Batch,*** **hpss_stage_batch_status_t** ∗ ***Status,*** **hpss_stage_batch_t** ∗ ***RetryBatch***)

Copy incomplete batch entries into a new retry batch.

Parameters

`in`	*Batch*	Already processed batch
`in`	*Status*	Parallel array of statuses for Batch
`out`	*RetryBatch*	New batch of retryable entries to retry

11.47.2.2 int API_GetBatchStatus (**hpss_stage_batch_status_t** ∗ ***Status,*** int ∗ ***Rc***)

Retrieve Batch Status.

Traverses a batch status structure and determines if any errors occurred.

Parameters

`in`	*Status*	Batch Status Structure
`in,out`	*Rc*	Batch Status Indicator

Return values

-EFAULT	NULL Status Structure
-EFAULT	NULL Return Code
-EINVAL	Invalid Status List Length
-EINVAL	NULL Status List Structure
0	Success

Note

The status of the first failure is returned, but there could be other failures subsequent to that one.

11.47.2.3 void API_StageBatchFree (hpss_stage_batch_t ∗ *Batch*)

Free a Batch Stage Request Structure.

Free a Batch Stage Request Structure

Parameters

`in,out`	*Batch*	Batch Request Structure

11.47.2.4 int API_StageBatchInit (hpss_stage_batch_t ∗ *Batch,* int *len*)

Initialize a Batch Stage Request Structure.

Create a Batch Stage Request Structure of the specified length

Parameters

`in,out`	*Batch*	Batch Request Structure
`in`	*len*	Batch Request Length

Return values

HPSS_E_NOERROR	Success
HPSS_EFAULT	NULL Batch Structure Provided
HPSS_ERANGE	The size of the batch is larger than the legal size. Split the batch into multiple batches of HPSS_MAX_STAGE_BATCH_LEN or smaller and try again.
HPSS_ENOMEM	Could not allocate batch structure

Note

The list structure is initialized with all zero values and the correct list length

11.47.2.5 void API_StageStatusFree (hpss_stage_batch_status_t ∗ *Status*)

Free a Batch Stage Status Structure.

Free a Batch Stage Status Structure

Parameters

`in,out`	*Status*	Batch Status Structure

11.47.2.6 int API_UpdateBatchStatus (hpss_stage_batch_t ∗ *LocalBatch,* hpss_stage_batch_t ∗ *FinalBatch,* hpss_stage_batch_status_t ∗ *LocalStatus,* hpss_stage_batch_status_t ∗ *FinalStatus*)

Update the batch after processing.

Parameters

`in`	*LocalBatch*	Local batch processing potential retries
`in`	*FinalBatch*	Final batch containing all requests
`out`	*LocalStatus*	Local batch status
`out`	*FinalStatus*	Final batch status

11.48 api_stat.c File Reference

Functions for retrieving a statistics structure for a file.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Fstat (int Fildes, hpss_stat_t ∗Buf)

  Retrieve statistics for an open file.
- int hpss_GetJunctionAttrs (const char ∗Path, hpss_Attrs_t ∗AttrOut)

  Retrieve file statistics associated with directory listing operations including junctions.
- int hpss_GetListAttrs (const char ∗Path, hpss_Attrs_t ∗AttrOut)

  Retrieve file statistics associated with directory listing operations.
- int hpss_Lstat (const char ∗Path, hpss_stat_t ∗Buf)

  Retrieve file, or link, statistics.
- int hpss_Stat (const char ∗Path, hpss_stat_t ∗Buf)

  Retrieve file statistics.

11.48.1 Detailed Description

Functions for retrieving a statistics structure for a file.

11.49 api_statfs.c File Reference

Functions for retrieving file system statistics.

```
#include "hpss_api.h"
#include "hpss_server_types.h"
#include "api_internal.h"
```

Functions

- int hpss_Statfs (uint32_t CosId, hpss_statfs_t ∗StatfsBuffer)

  Retrieve file system status information for a class of service.
- int hpss_Statfs64 (uint32_t CosId, hpss_statfs64_t ∗StatfsBuffer)

  Retrieve file system status information for a class of service.
- int hpss_Statvfs (uint32_t CosId, hpss_statvfs_t ∗StatvfsBuffer)

  Retrieve file system status information for a class of service in VFS format.
- int hpss_Statvfs64 (uint32_t CosId, hpss_statvfs64_t ∗StatvfsBuffer)

  Retrieve file system status information for a class of service in VFS format.

11.49.1 Detailed Description

Functions for retrieving file system statistics.

11.50 api_statsubsys.c File Reference

Functions for retrieving subsystem statistics.

```
#include <stdlib.h>
#include <pthread.h>
#include <api_internal.h>
```

Functions

- int hpss_GetSubSysLimits (uint32_t SubsystemID, const sec_cred_t ∗Ucred, hpss_subsys_limits_t ∗Limits-Out)

    Retrieve subsystem limits.
- int hpss_GetSubSysStats (uint32_t SubsystemID, subsys_stats_t ∗StatsOut)

    Retrieve subsystem statistics.
- int hpss_GetTrashSettings (ns_TrashcanSettings_t ∗TrashSettings)

    Query the Root Core Server for trashcan settings.
- int hpss_ResetSubSysStats (uint32_t SubsystemID, subsys_stats_t ∗StatsOut)

    Reset subsystem statistics.

11.50.1 Detailed Description

Functions for retrieving subsystem statistics.

11.51 api_stream.c File Reference

Functions for using stream-based I/O.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_Getenv.h"
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Fclose (HPSS_FILE ∗hpss_Stream)

    Closes a stream and cleans up.
- int hpss_Fcntl (int hpss_fd, int Cmd, long Arg)

    Set or get file control arguments.
- int hpss_Fflush (HPSS_FILE ∗hpss_Stream)

    Writes buffered data for a stream.

- int hpss_Fgetc (HPSS_FILE ∗stream)

  Retrieve a character from the stream.
- char ∗ hpss_Fgets (char ∗s, int n, HPSS_FILE ∗stream)

  Read a string from a stream buffer.
- HPSS_FILE ∗ hpss_Fopen (const char ∗Path, const char ∗Mode)

  Opens an HPSS file and associates a stream with it.
- size_t hpss_Fread (void ∗Ptr, size_t Size, size_t Num, HPSS_FILE ∗hpss_Stream)

  Provides a buffered I/O front-end to the hpss_Read function.
- int hpss_Fseek (HPSS_FILE ∗hpss_Stream, int64_t OffsetIn, int Whence)

  Seek to an offset within the stream.
- int hpss_Fsync (int hpss_fd)

  Checks that a file descriptor is valid.
- long hpss_Ftell (HPSS_FILE ∗hpss_Stream)

  Retrieve the current offset of the stream.
- size_t hpss_Fwrite (const void ∗Ptr, size_t Size, size_t Num, HPSS_FILE ∗hpss_Stream)

  Provides a buffered I/O front-end to hpss_Write.

11.51.1 Detailed Description

Functions for using stream-based I/O.

11.52 api_symlink.c File Reference

Functions for manipulating symbolic links.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Symlink (const char ∗Contents, const char ∗Path)

  Create a symbolic link.
- int hpss_SymlinkHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Contents, const char ∗Path, const sec_cred_t ∗Ucred, hpss_vattr_t ∗AttrsOut)

  Create a symbolic link using a handle.

11.52.1 Detailed Description

Functions for manipulating symbolic links.

11.53 api_trunc.c File Reference

Functions for truncating file data.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Fclear (int Fildes, uint64_t Length)

  Create a hole in an open file at its current offset.
- int hpss_FclearOffset (int Fildes, uint64_t Offset, uint64_t Length)

  Create a hole in an open file at a specified offset.
- int hpss_Fpreallocate (int Fildes, uint64_t Length)

  Preallocate storage resources for an open file.
- int hpss_Ftruncate (int Fildes, uint64_t Length)

  Truncate an open file.
- int hpss_Truncate (const char ∗Path, uint64_t Length)

  Truncate a file.
- int hpss_TruncateHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, uint64_t Length, const sec-_cred_t ∗Ucred)

  Truncate a file using a handle.

11.53.1 Detailed Description

Functions for truncating file data.

11.54 api_umask.c File Reference

Functions for manipulating the file creation mask.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- mode_t hpss_Umask (mode_t CMask)

  Set the file mode creation mask, and returns the previous mask value.

11.54.1 Detailed Description

Functions for manipulating the file creation mask.

11.55 api_undelete.c File Reference

Functions for undeleting HPSS files.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "u_signed64.h"
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Undelete (const char ∗Path, hpss_undelete_flags_t Flag)

  Restore a namespace entry from the trash.
- int hpss_UndeleteHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, hpss_undelete_flags_t Flag)

  Restore a namespace entry from the trash.

11.55.1 Detailed Description

Functions for undeleting HPSS files.

11.56 api_unlink.c File Reference

Functions for unlinking HPSS files.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "u_signed64.h"
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int Common_hpss_Unlink (const char ∗Path, int Immediate)

  Remove a namespace entry, optionally bypassing the trashcan.
- int Common_hpss_UnlinkHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, int Immediate)

  Remove a namespace entry using a handle, bypassing the trashcan.
- int hpss_Unlink (const char ∗Path)

  Remove a namespace entry.
- int hpss_UnlinkHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Remove a namespace entry using a handle.
- int hpss_UnlinkHandleImmediate (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Remove a namespace entry using a handle, bypassing the trashcan.

- int hpss_UnlinkImmediate (const char ∗Path)

  Remove a namespace entry, bypassing the trashcan.

11.56.1 Detailed Description

Functions for unlinking HPSS files.

11.56.2 Function Documentation

11.56.2.1 int Common_hpss_Unlink (const char ∗ *Path,* int *Immediate*)

Remove a namespace entry, optionally bypassing the trashcan.

The 'Common_hpss_Unlink' function removes the entry name by 'Path', and decrements the link count for the file. If link count becomes zero, the file will be deleted when it is no longer open to any client. It allows the specification of whether to bypass the trashcan or not.

Parameters

in	*Path*	Path of file to unlink
in	*Immediate*	Bypass the trashcan

Return values

0	Unlink was successful.
-EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
-EFAULT	The Path parameter is a NULL pointer.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The named file does not exist, or the Path argument points to an empty string.
-ENOTDIR	A component of the Path prefix is not a directory.
-EPERM	The file named by Path is a directory.

11.57 api_userattr.c File Reference

Functions for accessing and manipulating User-defined Attributes.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <ctype.h>
#include "hpss_api.h"
#include "api_internal.h"
#include "hpss_xml.h"
```

Functions

- int hpss_UserAttrCloseCursor (int SubsystemId, const hpss_cursor_id_t ∗CursorId)

  Close a cursor.

- int hpss_UserAttrDeleteAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, const hpss_userattr_list_t ∗Attrs, const char ∗Schema)

  Delete attributes from a namespace path using a handle.
- int hpss_UserAttrDeleteAttrs (const char ∗Path, const hpss_userattr_list_t ∗Attrs, const char ∗Schema)

  Delete User-defined Attributes on a namespace path.
- int hpss_UserAttrGetAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, hpss_userattr_list_t ∗Attrs, int XMLFlag, int XMLSize)

  Retrieve User-defined Attributes on a namespace path using a handle.
- int hpss_UserAttrGetAttrs (const char ∗Path, hpss_userattr_list_t ∗Attrs, int XMLFlag, int XMLSize)

  Retrieve User-defined Attributes on a namespace path.
- int hpss_UserAttrGetObj (int SubsystemId, const hpss_userattr_list_t ∗Attrs, int32_t MaxObjs, object_id_-hash_list_t ∗ObjectList, hpss_cursor_id_t ∗CursorId)

  Retrieve object id / hash pairs from a subsystem which match the attribute.
- int hpss_UserAttrGetObjQuery (int SubsystemId, int32_t MaxObjs, object_id_hash_list_t ∗ObjectList, const char ∗Query, hpss_cursor_id_t ∗CursorId)

  Retrieve object id / hash pairs which match an xquery.
- int hpss_UserAttrGetXML (int SubsystemId, int32_t MaxXMLs, hpss_string_list_t ∗XML, const char ∗XQuery, hpss_cursor_id_t ∗CursorId, int XMLSize)

  Retrieve XML strings from the subsystem which match the attribute.
- int hpss_UserAttrGetXMLObj (int SubsystemId, int32_t MaxXMLObjs, object_id_hash_list_t ∗ObjectList, hpss_string_list_t ∗XML, const char ∗XPathWhere, const char ∗XPathFind, hpss_cursor_id_t ∗CursorId, int XMLSize)

  Retrieve object / XML pairs whitch match the query.
- int hpss_UserAttrListAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, hpss_userattr_list_t ∗Attrs, int Flags, int XMLSize)

  List User-defined Attributes associated with a namespace path using a handle.
- int hpss_UserAttrListAttrs (const char ∗Path, hpss_userattr_list_t ∗Attrs, int Flags, int XMLSize)

  List User-defined Attributes associated with a namespace path.
- int hpss_UserAttrReadObj (int SubsystemId, object_id_hash_list_t ∗ObjectList, hpss_cursor_id_t ∗CursorId)

  Read objects from a previous object search.
- int hpss_UserAttrReadXML (int SubsystemId, hpss_string_list_t ∗XML, hpss_cursor_id_t ∗CursorId)

  Read object ids from a previous XML search.
- int hpss_UserAttrReadXMLObj (int SubsystemId, object_id_hash_list_t ∗ObjectList, hpss_string_list_t ∗XML, hpss_cursor_id_t ∗CursorId)

  Read more XML/objects from a previous XML/Object search.
- int hpss_UserAttrSetAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, const hpss_userattr_list_t ∗Attrs, const char ∗Schema)

  Set User-defined Attributes on a namespace path using a handle.
- int hpss_UserAttrSetAttrs (const char ∗Path, const hpss_userattr_list_t ∗Attrs, const char ∗Schema)

  Set User-defined Attributes on a namespace path.
- int hpss_UserAttrXQueryGet (const char ∗Path, const char ∗XQuery, char ∗Buffer, int XMLSize)

  Retrieve User-defined Attribute data for a path using XQuery.
- int hpss_UserAttrXQueryGetHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_-t ∗Ucred, const char ∗XQuery, char ∗Buffer, int XMLSize)

  Retrieve User-defined Attribute data for a path using XQuery and an object handle.
- int hpss_UserAttrXQueryUpdate (const char ∗Path, const char ∗XQuery, const char ∗Schema)

  Update User-defined Attributes on a path using an XQuery string.
- int hpss_UserAttrXQueryUpdateHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, const char ∗XQuery, const char ∗Schema)

  Update User-defined Attributes on a path using an XQuery string and a handle.

11.57.1 Detailed Description

Functions for accessing and manipulating User-defined Attributes.

11.58 api_utime.c File Reference

Functions for setting the access and modification time of a file.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hpss_api.h"
#include "api_internal.h"
```

Functions

- int hpss_Utime (const char ∗Path, const struct utimbuf ∗Times)

  Set the access and modification times of a file.

11.58.1 Detailed Description

Functions for setting the access and modification time of a file.

11.59 api_wrappers.c File Reference

Wrappers around Core Server Interfaces.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/select.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include "hpss_soid_func.h"
#include "hpss_api.h"
#include "api_internal.h"
#include "core_interface.h"
#include "hpss_varattrs_fn.h"
#include "acct_av_lib.h"
```

Functions

- int API_core_OpenBitfile (apithrdstate_t ∗ThreadContext, hpss_reqid_t RequestID, const sec_cred_t ∗Ucred, const bfs_bitfile_obj_handle_t ∗BitfileObj, uint32_t OpenFlags, hpss_cos_md_t ∗RetCOSPtr, hpss_sclass_-md_t ∗RetSClassPtr, hpss_object_handle_t ∗RetBFHandle)

Open an HPSS file by Bitfile Id.

11.59.1 Detailed Description

Wrappers around Core Server Interfaces.

11.59.2 Function Documentation

11.59.2.1 int API_core_OpenBitfile (apithrdstate_t ∗ *ThreadContext,* hpss_reqid_t *RequestID,* const sec_cred_t ∗ *Ucred,* const bfs_bitfile_obj_handle_t ∗ *BitfileObj,* uint32_t *OpenFlags,* hpss_cos_md_t ∗ *RetCOSPtr,* hpss_sclass_md_t ∗ *RetSClassPtr,* hpss_object_handle_t ∗ *RetBFHandle*)

Open an HPSS file by Bitfile Id.

Parameters

`in`	*ThreadContext*	Client Thread Context
`in`	*RequestID*	Request Id
`in`	*Ucred*	User Credentials
`in`	*BitfileObj*	HPSS Bitfile Object
`in`	*OpenFlags*	Flags
`out`	*RetCOSPtr*	COS Values
`out`	*RetSClassPtr*	Level 0 Storage Class
`out`	*RetBFHandle*	Bitfile Handle

Return values

0	Success.
Other	Error code indicating the nature of the problem

Note

None.

11.60 api_write.c File Reference

Functions for writing file data in HPSS.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include "hpss_pthread.h"
#include "hpss_api.h"
#include "hpss_memalign.h"
#include "api_internal.h"
#include "mvr_protocol.h"
#include "pdata.h"
#include "traniod.h"
#include "san3p.h"
#include "san3p_util.h"
#include "hpss_interop.h"
```

Functions

- ssize_t hpss_Write (int Fildes, const void ∗Buf, size_t Nbyte)

  Write data to an open file.

11.60.1 Detailed Description

Functions for writing file data in HPSS.

11.61 api_wrtlist.c File Reference

Functions for accessing the writelist interface.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <unistd.h>
#include <fcntl.h>
#include "hpss_pthread.h"
#include "hpss_api.h"
#include "api_internal.h"
#include "pdata.h"
#include "mvr_protocol.h"
#include "hpss_RequestID.h"
#include "hpss_netopt.h"
#include "hpsscfgx_prototypes.h"
#include "hpss_iod.h"
#include "hpss_interop.h"
#include "san3p.h"
#include "san3p_util.h"
```

Functions

- int hpss_WriteList (const hpss_IOD_t ∗IODPtr, uint32_t Flags, hpss_IOR_t ∗IORPtr)

  Issue a write IOD to HPSS.

11.61.1 Detailed Description

Functions for accessing the writelist interface.

11.61.2 Function Documentation

11.61.2.1 int hpss_WriteList (const hpss_IOD_t ∗ *IODPtr*, uint32_t *Flags*, hpss_IOR_t ∗ *IORPtr*)

Issue a write IOD to HPSS.

Parameters

`in`	*IODPtr*	IO description
`in`	*Flags*	IO specific flags
`out`	*IORPtr*	results of the IO

The 'hpss_WriteList' function writes the file data specified by the source descriptor list in the IOD pointed to by 'IODPtr' and moves the data to destinations specified by the sink descriptor list in the IOD. Results of the request will be returned in the structure pointed to by 'IORPtr'.

Return values

0	Success

Note

hpss_WriteList should not be directly used by client applications; it is meant to be used with internal HPSS client applications only.

11.62 bfs_userif_def.x File Reference

Client to bitfile interface types.

Classes

- struct bf_activity_count

  Activity counts for a bitfile. More...
- struct bf_sc_attrib
- struct bf_vv_attrib
- struct bf_xattrib
- struct bfs_bitfile_obj_handle
- struct bfs_callback_addr
- struct bfs_callback_ret_msg
- struct bfs_file_hash_digest

  This data structure is used to pass hash digest related information between the client and Core Server. More...
- struct hpss_stage
- struct hpss_stage_batch

- struct hpss_stage_batch_status
- struct hpss_stage_bitfile_list
- struct hpss_stage_status

Variables

- const int BFS_FILE_HASH_DIGEST_FROM_USER = 0x0001
- const int BFS_FILE_HASH_DIGEST_GENERATED = 0x0008
- const int BFS_FILE_HASH_DIGEST_INVALID = 0x0010
- const int BFS_FILE_HASH_DIGEST_SKIP_VERF = 0x0004
- const int BFS_FILE_HASH_DIGEST_VERIFIED = 0x0002
- const int BFS_MAX_VV_TO_RETURN_AT_LEVEL = 10

11.62.1 Detailed Description

Client to bitfile interface types.

11.62.2 Class Documentation

11.62.2.1 struct bf_activity_count

Activity counts for a bitfile.

Class Members

signed32	CopyFileCount	# active file copies (changecos), -1 is not set
signed32	MigrateCount	# active copy/move, -1 is not set
signed32	ModifyCount	# modify type requests such as purge, clear, -1 is not set
signed32	ReadCount	# active reads, -1 is not set
signed32	StageCount	# active stages, -1 is not set
signed32	WriteCount	# active writes, -1 is not set

11.62.2.2 struct bf_sc_attrib

Bitfile storage class metadata

Class Members

uint64_t	BytesAtLevel	Amount of data that exists at this level (in bytes)
uint32_t	Flags	Flags describing the state of the data. The flags that define the state of the fileset. Valid values include: • BFS_BFATTRS_LEVEL_IS_DISK This is a disk storage level. • BFS_BFATTRS_LEVEL_IS_TAPE This is a tape storage level. • BFS_BFATTRS_DATAEXISTS_AT_LEVEL Data exists at this level. • BFS_BFATTRS_ADDITIONAL_VV_EXIST More VVs exist than the structure can hold.
uint32_t	NumberOfVVs	Number of VV entries in the array
uint32_t	Optimum-AccessSize	Optimum access size of the storage class
uint64_t	StripeLength	Stripe length of the storage class
uint32_t	StripeWidth	Stripe width of the storage class
bf_vv_attrib_t	VVAttrib[BFS_-MAX_VV_TO_-RETURN_AT_L-EVEL]	Virtual volumes which contain bitfile segments

11.62.2.3 struct bf_vv_attrib

Bitfile virtual volume attributes

Class Members

uint64_t	BytesOnVV	Bytes stored on this VV. Specifies the number of bytes of the given file stored on this VV from the current contiguous set of bitfile segments. It is possible for BytesOnVV to be less than the total number of bytes of the given file that reside on this VV if non-adjacent bitfile segments are stored on the same VV. For example, if a file consists of three 1MB bitfile segments, with the first and third on VV 'A' and the second on VV 'B', an extended attributes call will return a list of three VVs, each with BytesOnVV = 1MB. On the other hand, if a file consists of three 1MB bitfile segments, with the first and second on VV 'A' and the third on VV 'B', an extended attributes call will return a list of two VVs, the first with BytesOnVV = 2MB and the second with BytesOnVV = 1MB.

pv_list_t ∗	PVList	List of physical volumes
int32_t	RelPosition	Relative start of the first segment on the VV
uint64_t	RelPosition-Offset	Relative position from the beginning of the aggregate
hpssoid_t	VVID	Virtual volume identifier

11.62.2.4 struct bf_xattrib

Volatile and metadata extended attributes

Class Members

bf_activity_-count_t	Activity	Current activity against the bitfile
byte	AllocMethod	Disk segment alloation method. Specifies the disk segment allocation method with which the bitfile was created. If the top level of the hierarchy is tape, this field is unused. See hpss_cos_md_t for an explanation of each allocation method.
bf_attrib_md_t	BfAttribMd	Bitfile metadata attributes
uint64_t	CurrentPosition	Specifies the current byte position in the bitfile.
uint32_t	FamilyId	Family identifier for the bitfile
uint64_t	MinSegSize	Minimum disk segment size. Specifies the minimum disk segment size last used to write the bitfile to top level of the hierarchy. If the top level of the hierarchy is tape, this field is unused and is always 0. If the top level of the hierarchy is disk and this field is 0, the file was created under a pre-7.1 version of HPSS and its minimum disk segment size at the top level of the hierarchy is the minimum size configured for the storage class at that level.
int32_t	OpenCount	Number of clients that have the bitfile open
bf_sc_attrib_t	SCAttrib[HPSS-_MAX_STORA-GE_LEVELS]	Storage class attributes at each valid hierarchy level

11.62.2.5 struct bfs_bitfile_obj_handle

Struture for passing BitfileID and Hash between BFS and clients

Class Members

hpss_-distributionkey_t	BfHash	Bitfile Hash value
hpssoid_t	BfId	Bitfile ID
uint32_t	ValidHash	Indicates BfHash has been set and should be correct. If off, BfHash is not available.

11.62.2.6 struct bfs_callback_addr

Async callback request type

Class Members

hpss_reqid_t	id	Id to be returned during the callback
hpss_sockaddr-_t	sockaddr	Host and port in network byte order

11.62.2.7 struct bfs_callback_ret_msg

Callback Return Message

Class Members

bfs_bitfile_obj_-handle_t	bfobj	Bitfile Id
hpss_reqid_t	reqid	request id
int32_t	retcode	return info

11.62.2.8 struct bfs_file_hash_digest

This data structure is used to pass hash digest related information between the client and Core Server.

Note

Do not confuse this with the bf_hash field of control data structure or the BfHash field of the bitfile object handle. The bf_hash and BfHash fields both hold database partitioning distribution keys.

Class Members

fstring	Creator[HPSS_-MAX_FILE_HA-SH_CREATOR-_NAME]	creator supplied text
opaque	Digest	file hash digest
uint16_t	Flags	file hash flags
timestamp_sec-_t	ModifyTime	Last modification time
hpss_hash_-type_t	Type	file hash type

11.62.2.9 struct hpss_stage

Stage Request Parameters

Class Members

bfs_bitfile_obj_-handle_t	BFObj	Bitfile Object
int	Filedes	Open File Decriptor (client only)
uint32_t	Flags	Stage request flags
uint64_t	FromStorage-Level	Storage Level to stage from

bfs_gk_ctl_t	GKControl	GK Control
uint64_t	Length	Length to stage, 0 stages all
hpss_object_-handle_t	ObjHandle	Object Handle
uint64_t	Offset	Offset to begin staging from
hpss_uuid_t *	RetryGKControl-NoP	Retry GK
uint32_t	ToStorageLevel	Storage Level to stage to

11.62.2.10 struct hpss_stage_batch

List of Stage Requests

11.62.2.11 struct hpss_stage_batch_status

List of Stage Request Status

11.62.2.12 struct hpss_stage_bitfile_list

List of Bitfile Containers

11.62.2.13 struct hpss_stage_status

Stage request status

Class Members

int32_t	Position	Stage Queue Position
hpss_reqid_t	RequestId	Request Id
int32_t	StageStatus	Stage Status

11.62.3 Variable Documentation

11.62.3.1 const int BFS_MAX_VV_TO_RETURN_AT_LEVEL = 10

Define the attributes of a specific storage class.

NOTE: It should be rare that a single file spans more than four tape VVs.

11.63 core_api_def.x File Reference

Interface definitions for the HPSS Core Server.

Classes

- struct hpss_core_read
- struct hpss_core_stage
- struct hpss_core_stage_callback

- struct hpss_read_queue_item
- struct hpss_read_queue_list
- struct hpss_reqid_list
- struct hpss_string_list

  String List conformant array. More...
- struct hpss_subsys_limits

  HPSS Subystem Limitations. More...
- struct object_id_hash
- struct object_id_hash_list

  Conformant array of object id hash pairs. More...
- struct object_id_list

  Conformant array of object ids. More...
- struct uda_string

  UDA string conformant array. More...

Enumerations

- enum hpss_stage_status_type_t { HPSS_STAGE_STATUS_BFS = 0, HPSS_STAGE_STATUS_BFS_SS = 1 }

  Level of detail for stage status.

Variables

- const int CORE_RQ_ONLY_NEW_READY = 0x000001
- const int HPSS_REMOVE_CONTEXT_ABORT = 0x00000001

11.63.1 Detailed Description

Interface definitions for the HPSS Core Server.

11.63.2 Class Documentation

11.63.2.1 struct hpss_core_read

Contains the required fields for an HPSS read operation

11.63.2.2 struct hpss_core_stage

Contains the required fields for an HPSS batch staging

11.63.2.3 struct hpss_core_stage_callback

Contains the required fields for an HPSS stage callback

11.63.2.4 struct hpss_read_queue_item

Contains the definition of an item for the read queue

11.63.2.5 struct hpss_read_queue_list

A list of read queue items

11.63.2.6 struct hpss_reqid_list

A list of request IDs

11.63.2.7 struct hpss_string_list

String List conformant array.

Has a member named StringList which is a structure containing:

- StringList_len - The length of the array
- StringList_val - The array of uda_string_t values

11.63.2.8 struct hpss_subsys_limits

HPSS Subystem Limitations.

Class Members

bool	AllowAny-Characters	Allow any characters
uint32_t	CurActiveCopyI-O	Current active copy requests
uint32_t	CurActiveIO	Current active I/O requests
uint32_t	CurConnections	Current connections
uint32_t	CurOpenBitfiles	Current open bitfiles
uint32_t	MaxActiveCopyI-O	Maximum active copy requests
uint32_t	MaxActiveIO	Maximum active I/O requests
uint32_t	MaxConnections	Max connections
uint32_t	MaxOpenBitfiles	Maximum open bitfiles
uint64_t	RequestQueue-Size	Number of entries in the read request queue
uint32_t	RequestQueue-Threads	Number of threads in the read request queue
uint32_t	TapeQueue-ActiveSet	Active set size
hpss_tape_-schedule_-method_t	TapeQueue-Method	Tape queue method
bool	Trashcan-Enabled	Trashcan enabled
uint32_t	TrashTime-UnlinkEligible	Trash eligibility time Number of seconds before a trashed file becomes eligible for permanent deletion.

uint32_t	UDAObjLimit	Max UDA search object limit
uint32_t	UDAObjXML-Limit	Max UDA search object + XML string limit
uint32_t	UDAXMLLimit	Max UDA search XML string limit
uint32_t	XMLRequest-Limit	Max XML (all queries) size
uint32_t	XMLSizeLimit	Max XML (single query) size

11.63.2.9 struct object_id_hash

Generic Object Id / Hash Pair

11.63.2.10 struct object_id_hash_list

Conformant array of object id hash pairs.

Has a member named ObjectList which is a structure containing:

- ObjectList_len - The length of the array
- ObjectList_val - The array of object_id_hash_t values

11.63.2.11 struct object_id_list

Conformant array of object ids.

Has a member named ObjectList which is a structure containing:

- ObjectList_len - The length of the array
- ObjectList_val - The array of uint64_t values

11.63.2.12 struct uda_string

UDA string conformant array.

Has a member named String which is a structure containing:

- String_len - The length of the array
- String_val - The array of string values

11.63.3 Enumeration Type Documentation

11.63.3.1 enum hpss_stage_status_type_t

Level of detail for stage status.

It should be noted that applications will incur additional cost in retrieving the SS stage status. Retrieving the SS stage status should be avoided if only the BFS stage status info is required.

Enumerator

HPSS_STAGE_STATUS_BFS Retrieve BFS stage status

HPSS_STAGE_STATUS_BFS_SS Retrieve BFS and SS stage status

11.63.4 Variable Documentation

11.63.4.1 const int CORE_RQ_ONLY_NEW_READY = 0x000001

Only return requests which have never previously reported ready

11.63.4.2 const int HPSS_REMOVE_CONTEXT_ABORT = 0x00000001

Abort requests

11.64 gk_gk_IFStructs.x File Reference

Define types pass through the Gatekeeper Services Interface.

Classes

- struct gk_OpenRequest

Enumerations

- enum gk_gk_RequestType { GK_INVALID_REQUEST_TYPE = 0, GK_CREATE_REQUEST = 1, GK_OPE-N_REQUEST = 2, GK_STAGE_REQUEST = 3 }

  Gatekeeper Request Types.
- enum gk_State {
  GK_INVALID_STATE = 0, GK_ERROR = 1, GK_GO = 2, GK_RETRY = 3,
  GK_SITE_CREATE = 4, GK_SITE_CREATE_STATS = 5, GK_SITE_CREATE_COMPLETE = 6, GK_SITE-_OPEN = 7,
  GK_SITE_OPEN_STATS = 8, GK_SITE_CLOSE = 9, GK_SITE_STAGE = 10, GK_SITE_STAGE_STATS = 11,
  GK_SITE_STAGE_COMPLETE = 12 }

  Gatekeeper entry states.

11.64.1 Detailed Description

Define types pass through the Gatekeeper Services Interface.

11.64.2 Class Documentation

11.64.2.1 struct gk_OpenRequest

Open Request Structure

Class Members

unsigned32	OFlag	Client Open Flags

11.64.3 Enumeration Type Documentation

11.64.3.1 enum gk_gk_RequestType

Gatekeeper Request Types.

Enumerator

GK_INVALID_REQUEST_TYPE	Invalid request
GK_CREATE_REQUEST	Create request
GK_OPEN_REQUEST	Open request
GK_STAGE_REQUEST	Stage request

11.64.3.2 enum gk_State

Gatekeeper entry states.

Enumerator

GK_INVALID_STATE	Invalid statel
GK_ERROR	Returning an error
GK_GO	Returning a good status
GK_RETRY	Returning a retry status
GK_SITE_CREATE	Calling gk_site_Create
GK_SITE_CREATE_STATS	Calling gk_site_CreateStats
GK_SITE_CREATE_COMPLETE	Calling gk_site_CreateComplete
GK_SITE_OPEN	Calling gk_site_Open
GK_SITE_OPEN_STATS	Calling gk_site_OpenStats
GK_SITE_CLOSE	Calling gk_site_Close
GK_SITE_STAGE	Calling gk_site_Stage
GK_SITE_STAGE_STATS	Calling gk_site_StageStats
GK_SITE_STAGE_COMPLETE	Calling gk_site_StageComplete

11.65 gk_RTM.h File Reference

```
#include "rtm_reqlist.h"
```

Classes

- struct gk_ReqInfo

  Gatekeeper RTM info. More...

Typedefs

- typedef struct gk_ReqInfo gk_ReqInfo_t

  Gatekeeper RTM info.

11.66 gk_SiteProtos.h File Reference

```
#include "gk_Entry.h"
```

Functions

- signed32 gk_site_Close (hpss_uuid_t ControlNo)

  Called when an HPSS file is being opened by an Authorized Caller.
- signed32 gk_site_CreateComplete (hpss_uuid_t ControlNo)

  Called when an HPSS file create has been completed.
- void gk_site_CreateStats (gk_EntryInfo_t EntryInfo)

  Called when an HPSS file is being created by an Authorized Caller.
- signed32 gk_site_GetMonitorTypes (u_signed64 ∗MonitorTypeBitsP)

  Called to get the request types being monitored.
- signed32 gk_site_Init (char ∗SitePolicyPathNameP)

  Called to initialize the Site Interface.
- signed32 gk_site_Open (gk_EntryInfo_t EntryInfo, unsigned32 ∗WaitTimeP)

  Called when an HPSS file is being opened.
- void gk_site_OpenStats (gk_EntryInfo_t EntryInfo)

  Called when an HPSS file is being opened by an Authorized Caller.
- signed32 gk_site_Stage (gk_EntryInfo_t EntryInfo, unsigned32 ∗WaitTimeP)

  Called when an HPSS file is being staged.
- signed32 gk_site_StageComplete (hpss_uuid_t ControlNo)

  Called when an HPSS file stage has completed.
- void gk_site_StageStats (gk_EntryInfo_t EntryInfo)

  Called when an HPSS file is being staged by an Authorized Caller. Authorzied Caller.

11.67 hacl.h File Reference

```
#include "hpss_api.h"
```

Classes

- struct hacl_acl_entry

  Defines an access control list entry. More...

Macros

- #define HACL_M_ALLOW_UNAUTHENTICATED (0x10000000)

  Allow unauthenticatied ACLs.
- #define HACL_M_REQUIRE_PERMS (0x40000000)

  Cause hacl_ConvertstringsToHACL to require permissions.
- #define HACL_M_USE_LOCAL_REALM_SPEC (0x20000000)

  Control explicit realm spec behavior in local ACLs.
- #define HACL_M_USE_MASK_OBJ (0x80000000)

  Print out effective permissions.
- #define HACL_MAX_ENTRY_SIZE

  Maximum HACL entry size.
- #define HACL_MAX_PERMS 8 /∗ rwxcid ∗/

  Maximum string size for permission string.
- #define HACL_MAX_TYPE 24 /∗ foreign_group_delegate ∗/

  Maximum string size for an ACL entry type.
- #define HPSS_EHACL_UNAUTH (-10500)

  Informative error indicating an authentication ACL entry.

Typedefs

- typedef struct hacl_acl_entry hacl_acl_entry_t

  Defines an access control list entry.

Functions

- int hacl_ConvertACLToHACL (ns_ACLConfArray_t ∗ACL_n,hacl_acl_t ∗∗ACL_h)

  Convert ACL to string format.
- int hacl_ConvertHACLPermsToPerms (char ∗HPerms,unsigned char ∗Perms)

  Convert permission string to HPSS format.
- int hacl_ConvertHACLToACL (hacl_acl_t ∗ACL_h,ns_ACLConfArray_t ∗ACL_n)

  Convert ACL to nameserver format.
- int hacl_ConvertHACLToString (hacl_acl_t ∗ACL_h,unsigned Flags,char const ∗EntrySeparator,char ∗∗ACL-_s)

  Convert ACL to a form suitable for printing.
- int hacl_ConvertHACLTypeToType (char ∗HType,unsigned char ∗Type)

  Convert ACL entry type to HPSS format.
- int hacl_ConvertPermsToHACLPerms (unsigned char Perms,char ∗HPerms)

  Convert HPSS permission mask to string.
- int hacl_ConvertStringsToHACL (int NumEntries,char ∗Entries[],char const ∗DefaultRealm,char const ∗WhiteSpaceChars,unsigned Flags,hacl_acl_t ∗∗ACL_h)

  Convert strings to HACL format.
- int hacl_ConvertTypeToHACLType (unsigned char Type,char ∗HType)

  Convert ACL entry type to string.
- int hacl_DeleteHACL (char ∗Path,unsigned32 Options,hacl_acl_t ∗ACL_h)

  Deletes selected ACL entries from the named object.
- int hacl_GetHACL (char ∗Path,unsigned32 Options,hacl_acl_t ∗∗ACL_h)

  Query HPSS to get the ACL for the named object.
- int hacl_SetHACL (char ∗Path,unsigned32 Options,hacl_acl_t ∗ACL_h)

  Replace the ACL of the named object with a new ACL.

- void hacl_SortHACL (hacl_acl_t ∗ACL_h)

  Sort an ACL into canonical order.
- int hacl_UpdateHACL (char ∗Path,unsigned32 Options,hacl_acl_t ∗ACL_h)

  Change the selected ACL entries for the named object.

11.68 hpss_api.h File Reference

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <stdlib.h>
#include <limits.h>
#include <utime.h>
#include <time.h>
#include <netdb.h>
#include <netinet/in.h>
#include "hpss_types.h"
#include "hpss_ls.h"
#include "hpss_pthread.h"
#include "hpss_net.h"
#include "hpss_sec_api.h"
#include "core_api_def.h"
#include "hpss_api_def.h"
#include "hpss_memalign.h"
#include "hpss_dirent.h"
#include "hpss_acl_cons.h"
#include "ns_Constants.h"
#include "hpss_obj_handle.h"
#include "hpss_soid_func.h"
#include "bfs_userif_def.h"
#include "hpss_iod_clnt.h"
#include "hpss_cos.h"
#include "hpss_limits.h"
#include "hpss_stat.h"
#include "hpss_vattr.h"
#include "hpss_RequestID.h"
#include "ss_pvlist.h"
#include "hpss_conv.h"
#include "u_signed64.h"
```

Classes

- struct api_config

  Controls optional features of the Client API configuration. More...
- struct api_dist_file_info_t

  Distributed file information structure. More...
- struct api_namespec

  Name specifier information. More...
- struct api_dist_file_info_t::file_info

  Information about the file entry. More...

- struct hpss_errno_state

  HPSS Error Code State. More...
- struct hpss_file_hash_digest

  Defined the input/output structure for hpss_FgetFileDigest and hpss_FsetFileDigest and hpss_FgetFileDigestList and hpss_FsetFileDigestList. More...
- struct hpss_file_hash_digest_list

  A list of hash digests for single or multi stripe digest handling. More...
- struct hpss_global_fsent_t

  Global fileset entry. More...
- struct hpss_junction_ent_t

  HPSS junction entry Structure to hold junction information. More...
- struct hpss_pio_gapinfo_s

  PIO sparse file gap information. More...
- struct hpss_pio_params_s

  PIO I/O parameters. More...
- struct hpss_userattr

  User-defined Attribute pair. More...
- struct hpss_userattr_list

  List of User-defined attribute pairs. More...
- struct subsys_stats

  HPSS Subystem statistics. More...

Macros

- #define API_CONFIG_ALL_FLAGS (0x0000001F)
- #define API_CONFIG_DEFAULT (0x00000000)
- #define API_DEBUG_ERROR (1)
- #define API_DEBUG_NONE (0)
- #define API_DEBUG_REQUEST (2)
- #define API_DEBUG_TRACE (4)
- #define API_DISABLE_CROSS_REALM (0x00000002)
- #define API_DISABLE_JUNCTIONS (0x00000004)
- #define API_ENABLE_LOGGING (0x00000001)
- #define API_MSGTYPE_MVRPROT MVRPROTOCOL_MSG_TYPE
- #define API_MSGTYPE_PDATA PDATA_MSG_TYPE
- #define API_TRANSFER_MVRSELECT 1
- #define API_TRANSFER_TCP 0
- #define API_USE_CONFIG (0x00000008)
- #define API_USE_SAN3P (0x00000010)
- #define HAVE_FSETFILEDIGEST 1

  HPSS file hash.
- #define HPSS_MIGRATE_FORCE (BFS_MIGRATE_FORCE)
- #define HPSS_MIGRATE_HPSS_ONLY (0x1)
- #define HPSS_MIGRATE_NO_COPY (BFS_MIGRATE_NO_COPY)
- #define HPSS_MIGRATE_PURGE_DATA (BFS_MIGRATE_PURGE_DATA)
- #define HPSS_O_EXCL (0x20000000)

  Open for exclusive access.
- #define HPSS_O_NO_TAPE (0x10000000)
- #define HPSS_O_STAGE_ASYNC (0x80000000)
- #define HPSS_O_STAGE_BKGRD (0x40000000)
- #define HPSS_O_STAGE_NONE (O_NONBLOCK)
- #define HPSS_PIO_HANDLE_GAP (1<<1)

- #define HPSS_PIO_PORT_RANGE (1<<2)
- #define HPSS_PIO_PUSH (1<<0)
- #define HPSS_PIO_USE_API_HOSTNAME (1<<3)
- #define HPSS_STAGE_STATUS_ACTIVE (2)
- #define HPSS_STAGE_STATUS_QUEUED (1)
- #define HPSS_STAGE_STATUS_UNKNOWN (0)
- #define MAX_XPATH_ELEM_SIZE (1001)
- #define MAX_XPATH_SIZE (MAX_XPATH_ELEM_SIZE∗125)
- #define UDA_API_VALUE 0
- #define UDA_API_XML 1
- #define XML_ATTR 1
- #define XML_NO_ATTR 2

Typedefs

- typedef struct api_config api_config_t

  Controls optional features of the Client API configuration.
- typedef struct api_namespec api_namespec_t

  Name specifier information.
- typedef struct hpss_errno_state hpss_errno_state_t

  HPSS Error Code State.
- typedef struct __HPSS_FILE HPSS_FILE
- typedef enum hpss_file_hash_flags hpss_file_hash_flags_t

  Defines the file hash flag values.
- typedef enum
  hpss_file_hash_stripe_flags hpss_file_hash_stripe_flags_t

  Preferencing flags for digest list operations.
- typedef int(∗ hpss_pio_cb_t)(void ∗, uint64_t, unsigned int ∗, void ∗∗)

  Parallel IO callback.
- typedef struct hpss_pio_gapinfo_s hpss_pio_gapinfo_t

  PIO sparse file gap information.
- typedef void ∗ hpss_pio_grp_t

  Parallel IO group handle.
- typedef uint32_t hpss_pio_options_t
- typedef struct hpss_pio_params_s hpss_pio_params_t

  PIO I/O parameters.
- typedef struct hpss_userattr_list hpss_userattr_list_t

  List of User-defined attribute pairs.
- typedef struct hpss_userattr hpss_userattr_t

  User-defined Attribute pair.
- typedef struct subsys_stats subsys_stats_t

  HPSS Subystem statistics.

Enumerations

- enum hpss_file_hash_flags {
  HPSS_FILE_HASH_DIGEST_VALID = (1<<0), HPSS_FILE_HASH_SET_BY_USER = (1<<1), HPSS_FI-LE_HASH_MIGRATE_VERIFIED = (1<<2), HPSS_FILE_HASH_SKIP_VERIFICATION = (1<<3), HPSS_FILE_HASH_GENERATED = (1<<4), HPSS_FILE_HASH_DIGEST_ACTIVE = (1<<5) }

  Defines the file hash flag values.

- enum hpss_file_hash_stripe_flags { HPSS_HASH_SINGLE_ONLY, HPSS_HASH_SINGLE, HPSS_HASH_-STRIPE, HPSS_HASH_STRIPE_ONLY }

  Preferencing flags for digest list operations.
- enum hpss_pio_operation_t { HPSS_PIO_READ, HPSS_PIO_WRITE }
- enum hpss_pio_transport_t { HPSS_PIO_TCPIP, HPSS_PIO_MVR_SELECT }

  Parallel IO transport types.
- enum hpss_readdir_flags_t { HPSS_READDIR_NONE = 0, HPSS_READDIR_GETATTRS = (1 << 0), HP-SS_READDIR_RAW = (1 << 1), HPSS_READDIR_NFS = (1 << 2) }
- enum hpss_undelete_flags_t { HPSS_UNDELETE_NONE = 0, HPSS_UNDELETE_RESTORE_TIME, HPS-S_UNDELETE_OVERWRITE, HPSS_UNDELETE_OVERWRITE_AND_RESTORE }
- enum namespec_type_t { NAMESPEC_SKIP, NAMESPEC_REALM, NAMESPEC_USER, NAMESPEC_G-ROUP }

  Name specifier translation types.
- enum notrunc_flag_t { NOTRUNC_CLEAR = 0, NOTRUNC_SET }

  Truncation flags.
- enum purgelock_flag_t { PURGE_UNLOCK = 0, PURGE_LOCK, SUPER_PURGE_UNLOCK, SUPER_PU-RGE_LOCK }

  Purge lock flags Enumerate the different types of hpss_PurgeLock flag settings.
- enum purgeonmigrate_flag_t { PURGE_ON_MIGRATE_CLEAR = 0, PURGE_ON_MIGRATE_SET }

  Purge on Migrate flags.

Functions

- uint64_t API_AddAllRegisterValues (int LastPosition)

  Set all register bit positions up to the position provided.
- u_signed64 API_AddRegisterValues (uint64_t InitialValue,...)

  Add values to a bit register.
- void API_CalcBlock32 (const hpss_Attrs_t ∗Attrs, uint32_t ∗Blksize, uint32_t ∗Numblks)

  Calculates the block size and number of blocks necessary for data.
- void API_CalcBlock64 (const hpss_Attrs_t ∗Attrs, uint64_t ∗Blksize, uint64_t ∗Numblks)

  Calculates the block size and number of blocks necessary for data.
- void API_ConvertModeToPosixMode (const hpss_Attrs_t ∗Attrs, mode_t ∗PosixMode)

  Converts an input HPSS mode value to its corresponding POSIX mode value.
- void API_ConvertPosixModeToMode (mode_t PosixMode, hpss_Attrs_t ∗Attrs)

  Converts a POSIX mode value to its equivalent HPSS mode value.
- void API_ConvertPosixTimeToTime (hpss_Attrs_t ∗Attrs, timestamp_sec_t Atime, timestamp_sec_t Mtime, timestamp_sec_t Ctime)

  Converts POSIX time values to corresponding HPSS time values.
- void API_ConvertTimeToPosixTime (const hpss_Attrs_t ∗Attrs, timestamp_sec_t ∗Atime, timestamp_sec_t ∗Mtime, timestamp_sec_t ∗Ctime)

  Converts HPSS time values to corresponding POSIX mode values.
- void API_FreeIOR (hpss_IOR_t ∗IORPtr)

  Frees memory that was allocated to an IOR.
- int API_GetBatchStatus (hpss_stage_batch_status_t ∗Status, int ∗Rc)

  Retrieve Batch Status.
- int API_GetClientAddr (const iod_srcsinkdesc_t ∗SrcSinkPtr, const iod_srcsinkreply_t ∗SrcSinkReplyPtr, uint64_t Offset, iod_address_t ∗∗RetAddr, uint64_t ∗RetLength, iod_srcsinkdesc_t ∗∗RetDescPtr, iod_-srcsinkreply_t ∗∗RetReplyPtr)

  Retrieves the client address for a data transfer.
- int API_GetDataThreadStackSize (void)

  Retrieves the data thread stack size.
- int API_GetServerIDForSubsystem (uint32_t SubsystemID, hpss_reqid_t RequestID, hpss_srvr_id_t ∗RetID)

Retrieves the server id of the core server of a given subsystem.
- hpss_reqid_t API_GetUniqueRequestID ()

  Retrieves a unique request id.
- uint64_t API_RemoveRegisterValues (uint64_t InitialValue,...)

  Remove values from a bit register.
- void API_StageBatchFree (hpss_stage_batch_t ∗Batch)

  Free a Batch Stage Request Structure.
- int API_StageBatchGetEntry (hpss_stage_batch_t ∗Batch, int Idx, hpss_stage_t ∗Entry)

  Retrieve Stage Batch Entry.
- int API_StageBatchInit (hpss_stage_batch_t ∗Batch, int len)

  Initialize a Batch Stage Request Structure.
- int API_StageBatchInsertBFHandle (hpss_stage_batch_t ∗Batch, int Idx, hpss_object_handle_t ∗ObjHandle, int FromStorageLevel, int ToStorageLevel, u_signed64 Offset, u_signed64 Length, uint32_t Flags)

  Set a Stage Batch Entry using a Bitfile Handle.
- int API_StageBatchInsertBFObj (hpss_stage_batch_t ∗Batch, int Idx, const bfs_bitfile_obj_handle_t ∗BfObj, int FromStorageLevel, int ToStorageLevel, u_signed64 Offset, u_signed64 Length, uint32_t Flags)

  Set a Stage Batch Entry using a Bitfile ID.
- int API_StageBatchInsertFd (hpss_stage_batch_t ∗Batch, int Idx, int Fd, int FromStorageLevel, int ToStorage-Level, u_signed64 Offset, u_signed64 Length, uint32_t Flags)

  Set a Stage Batch Entry using a File Descriptor.
- void API_StageStatusFree (hpss_stage_batch_status_t ∗Status)

  Free a Batch Stage Status Structure.
- int hpss_AbortIOByMoverId (int32_t SubsystemId, hpss_srvr_id_t MoverId)

  Abort all I/O requests for the specified mover ID.
- int hpss_AbortIOByRequestId (int32_t SubsystemId, hpss_reqid_t RequestIdToAbort, sec_cred_t ∗Ucred)

  Abort a specific I/O request.
- int hpss_AbortIOByRequestMatch (int32_t SubsystemId, char ∗PartialReqString, sec_cred_t ∗Ucred)

  Abort an I/O request that matches the request string.
- int hpss_AbortIOByUserId (int32_t SubsystemId, sec_cred_t ∗Ucred, uint32_t ∗RequestsAborted, uint32_t ∗RequestsAlreadyAborted, uint32_t ∗RequestsFound)

  Abort all I/O requests for the specified user.
- int hpss_Access (const char ∗Path, int Amode)

  Checks the accessibility of a file.
- int hpss_AccessHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, int Amode, const sec_cred_t ∗Ucred)

  Checks the accessibility of a file using a handle.
- int hpss_AcctCodeToName (acct_rec_t AcctCode, hpss_id_t ∗Site, char ∗AcctName)

  Maps an account code to its corresponding account name.
- int hpss_AcctNameToCode (char ∗AcctName, hpss_id_t ∗Site, acct_rec_t ∗AcctCode)

  Maps an account name to its corresponding account code.
- int hpss_Chacct (const char ∗Path, acct_rec_t AcctCode)

  Changes a file or directory's account code.
- int hpss_ChacctByName (const char ∗Path, const char ∗AcctName)

  Changes a file or directory's account name.
- int hpss_Chdir (const char ∗Path)

  Changes the current working directory.
- int hpss_Chmod (const char ∗Path, mode_t Mode)

  Alters a file or directory's permissions.
- int hpss_Chown (const char ∗Path, uid_t Owner, gid_t Group)

  Changes the user id and group id of a file or directory.
- int hpss_Chroot (const char ∗Path)

Sets the client's root directory.

- void hpss_ClearLastHPSSErrno (void)

  Clears the current HPSS errno state.
- int hpss_ClientAPIInit (void)

  Initialize the Client API for the current thread.
- void hpss_ClientAPIReset (void)

  Resets the client API state.
- int hpss_Close (int Fildes)

  Terminates the connection between a file and its handle.
- int hpss_Closedir (int Dirdes)

  Close an open directory stream.
- int hpss_ConvertIdsToNames (int32_t NumEntries, api_namespec_t ∗Specs)

  Converts user or group ids to names.
- int hpss_ConvertNamesToIds (int32_t NumEntries, api_namespec_t ∗Specs)

  Converts user or group names to ids.
- int hpss_CopyFile (int SrcFildes, int DestFildes)

  Copies the source file to the destination file.
- int hpss_Creat (const char ∗Path, mode_t Mode, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

  Create a file and open it.
- int hpss_Create (const char ∗Path, mode_t Mode, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

  Create an HPSS file.
- int hpss_CreateHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, mode_t Mode, sec_cred_t ∗Ucred, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut, hpss_vattr_t ∗AttrsOut)

  Create a file using a handle.
- int hpss_DeleteACL (const char ∗Path, uint32_t Options, const ns_ACLConfArray_t ∗ACL)

  Removes an ACL entry from a file or directory.
- int hpss_DeleteACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t Options, const ns_ACLConfArray_t ∗ACL)

  Removes an ACL entry from a file or directory using a handle.
- int hpss_Fclear (int Fildes, uint64_t Length)

  Create a hole in an open file at its current offset.
- int hpss_FclearOffset (int Fildes, uint64_t Offset, uint64_t Length)

  Create a hole in an open file at a specified offset.
- int hpss_Fclose (HPSS_FILE ∗hpss_Stream)

  Closes a stream and cleans up.
- int hpss_Fcntl (int hpss_id, int cmd, long arg)

  Set or get file control arguments.
- int hpss_FdelFileDigestList (int Fildes, hpss_file_hash_stripe_flags_t Flags)

  Delete file hash digest information.
- int hpss_Fflush (HPSS_FILE ∗hpss_Stream)

  Writes buffered data for a stream.
- int hpss_Fgetc (HPSS_FILE ∗stream)

  Retrieve a character from the stream.
- int hpss_FgetFileDigest (int Fildes, hpss_file_hash_digest_t ∗Digest)

  Retrieves the file's hash digest, if any, for an open file.
- int hpss_FgetFileDigestList (int Fildes, hpss_file_hash_stripe_flags_t Flags, uint64_t ∗StripeLength, hpss_file_hash_digest_list_t ∗DigestList)

  Retrieves a file's hash digest, if any, for an open file.

- char ∗ hpss_Fgets (char ∗s, int n, HPSS_FILE ∗stream)

    Read a string from a stream buffer.
- int hpss_FileGetAttributes (const char ∗Path, hpss_fileattr_t ∗AttrOut)

    Queries the attributes of a file.
- int hpss_FileGetAttributesBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, char ∗Path, hpss_fileattr_t ∗Attr-Out)

    Get file attributes using a Bitfile Object.
- int hpss_FileGetAttributesHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, hpss_fileattr_t ∗AttrOut)

    Queries the attributes of a file using a handle.
- int hpss_FileGetDigestHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, sec_cred_t ∗Ucred, hpss_file_hash_digest_t ∗Digest)

    Retrieves the file's hash digest, if any.
- int hpss_FileGetDigestListHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, sec_cred_t ∗Ucred, hpss_file_hash_stripe_flags_t Flags, uint64_t ∗StripeLength, hpss_file_hash_digest_list_t ∗Digest)

    Retrieves a file's hash digest(s) by preference, if any.
- int hpss_FileGetXAttributes (const char ∗Path, uint32_t Flags, uint32_t StorageLevel, hpss_xfileattr_t ∗Attr-Out)

    Queries the attributes of a file using flags and storage level perms.
- int hpss_FileGetXAttributesHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t Flags, uint32_t StorageLevel, hpss_xfileattr_t ∗AttrOut)

    Queries the attributes of a file using a handle, flags, and a storage level.
- int hpss_FileSetAttributes (const char ∗Path, hpss_fileattrbits_t SelFlags, const hpss_fileattr_t ∗AttrIn, hpss-_fileattr_t ∗AttrOut)

    Changes the attributes of an object.
- int hpss_FileSetAttributesBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, hpss_fileattrbits_t SelFlags, const hpss_fileattr_t ∗AttrIn, hpss_fileattr_t ∗AttrOut, char ∗Path)

    Change the attributes of an object specified by bitfile object.
- int hpss_FileSetAttributesHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, hpss_fileattrbits_t SelFlags, const hpss_fileattr_t ∗AttrIn, hpss_fileattr_t ∗AttrOut)

    Changes the attributes of an object based upon an object handle and path.
- int hpss_FileSetCOS (const char ∗Path, uint32_t COSId, uint32_t StreamId)

    Change the COS of an file.
- int hpss_FileSetCOSHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t COSId, uint32_t StreamId)

    Change the COS of a file.
- int hpss_FilesetCreate (const hpss_srvr_id_t ∗CoreServerID, uint32_t CreateOptions, ns_FilesetAttrBits-_t FilesetAttrBits, const ns_FilesetAttrs_t ∗FilesetAttrs, hpss_AttrBits_t ObjectAttrBits, const hpss_Attrs_-t ∗ObjectAttrs, ns_FilesetAttrBits_t RetFilesetAttrBits, hpss_AttrBits_t RetObjectAttrBits, ns_FilesetAttrs_t ∗RetFilesetAttrs, hpss_Attrs_t ∗RetObjectAttrs, ns_ObjHandle_t ∗FilesetHandle)

    Creates a new fileset.
- int hpss_FilesetDelete (const char ∗Name, const uint64_t ∗FilesetId, const ns_ObjHandle_t ∗FilesetHandle)

    Deletes an existing fileset.
- int hpss_FilesetGetAttributes (const char ∗Name, const uint64_t ∗FilesetId, const ns_ObjHandle_t ∗Fileset-Handle, const hpss_srvr_id_t ∗CoreServerID, ns_FilesetAttrBits_t FilesetAttrBits, ns_FilesetAttrs_t ∗Fileset-Attrs)

    Retrieves the attributes of a fileset.
- int hpss_FilesetListAll (uint64_t OffsetIn, uint32_t Entries, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_global_-fsent_t ∗FSentPtr)

    Retrieves attributes for every HPSS fileset.
- int hpss_FilesetSetAttributes (const char ∗Name, const uint64_t ∗FilesetId, const ns_ObjHandle_t ∗Fileset-Handle, ns_FilesetAttrBits_t FilesetAttrBitsIn, const ns_FilesetAttrs_t ∗FilesetAttrsIn, ns_FilesetAttrBits_-t FilesetAttrBitsOut, ns_FilesetAttrs_t ∗FilesetAttrsOut)

Sets the attributes of a fileset.

- HPSS_FILE ∗ hpss_Fopen (const char ∗Path, const char ∗Mode)

  Opens an HPSS file and associates a stream with it.
- int hpss_Fpreallocate (int Fildes, uint64_t Length)

  Preallocate storage resources for an open file.
- size_t hpss_Fread (void ∗Ptr, size_t Size, size_t Num, HPSS_FILE ∗hpss_Stream)

  Provides a buffered I/O front-end to the hpss_Read function.
- int hpss_Fseek (HPSS_FILE ∗hpss_Stream, int64_t OffsetIn, int Whence)

  Seek to an offset within the stream.
- int hpss_FsetFileDigest (int Fildes, const hpss_file_hash_digest_t ∗Digest)

  Sets single stripe file's hash digest information.
- int hpss_FsetFileDigestList (int Fildes, uint64_t StripeLength, const hpss_file_hash_digest_list_t ∗Digest)

  Sets file hash digest information.
- int hpss_Fstat (int Fildes, hpss_stat_t ∗Buf)

  Retrieve statistics for an open file.
- int hpss_Fsync (int hpss_fd)

  Checks that a file descriptor is valid.
- long hpss_Ftell (HPSS_FILE ∗hpss_Stream)

  Retrieve the current offset of the stream.
- int hpss_Ftruncate (int Fildes, uint64_t Length)

  Truncate an open file.
- size_t hpss_Fwrite (const void ∗Ptr, size_t Size, size_t Num, HPSS_FILE ∗hpss_Stream)

  Provides a buffered I/O front-end to hpss_Write.
- int hpss_GetAcct (acct_rec_t ∗RetDefAcct, acct_rec_t ∗RetCurAcct)

  Retrieves current and default account codes.
- int hpss_GetAcctName (char ∗AcctName)

  Retrieves account name.
- int hpss_GetACL (const char ∗Path, uint32_t Options, ns_ACLConfArray_t ∗ACL)

  Retrives the ACL of a file or directory.
- int hpss_GetACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t Options, ns_ACLConfArray_t ∗ACL)

  Retrieves an ACL of a file or directory using a handle.
- int hpss_GetAllSubsystems (ls_map_array_t ∗∗ls_map_array)

  Retrieve a map of all subsystems.
- int hpss_GetAsyncStatus (hpss_reqid_t CallBackId, bfs_bitfile_obj_handle_t ∗BitfileObj, int32_t ∗Status)

  Get the status of a background stage.
- int hpss_GetAttrHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗HandleOut, hpss_vattr_t ∗AttrOut)

  Obtains information about a file or directory using a handle.
- int hpss_GetBatchAsynchStatus (hpss_reqid_t CallBackId, hpss_stage_bitfile_list_t ∗BFIDs, hpss_stage_-status_type_t Type, hpss_stage_batch_status_t ∗Status)

  Check the status of files in a batch request.
- int hpss_GetConfiguration (api_config_t ∗ConfigOut)

  Retrieves the current values used for default configurations.
- int hpss_Getcwd (char ∗Buf, size_t Size)

  Retrieves the current working directory.
- int hpss_Getdents (const ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_t ∗Ucred, uint32_t BufferSize, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

  Retrieve directory entries with reliable entry offsets.
- int hpss_GetdentsAttrs (ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_t ∗Ucred, uint32_t BufferSize, uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

Read directory entries and object attributes in an offset-reliable manner.

- int hpss_GetDistFile (int Fildes, api_dist_file_info_t ∗FileInfo)

  Extracts a file table entry for an open file descriptor.
- int hpss_GetFileNotrunc (const char ∗Path, int ∗NotruncFlag)

  Retrieves the file's NOTRUNC_FINAL_SEG flag.
- int hpss_GetFullPath (const ns_ObjHandle_t ∗ObjHandle, char ∗∗FullPath)

  Provides the full path back to an object from the root of roots.
- int hpss_GetFullPathBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObject, char ∗∗FullPath)

  Provides a full path back to a bitfile from the root of roots.
- int hpss_GetFullPathBitfileLen (const bfs_bitfile_obj_handle_t ∗BitfileObject, char ∗FullPath, size_t BufLen)

  Provides a full path back to a bitfile from the root of roots.
- int hpss_GetFullPathHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, char ∗∗FullPath)

  Provides the full path back to an object from the root of roots.
- int hpss_GetFullPathHandleLen (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, char ∗FullPath, size_t BufLen)

  Provides the full path back to an object from the root of roots.
- int hpss_GetFullPathLen (const ns_ObjHandle_t ∗ObjHandle, char ∗Path, size_t BufLen)

  Provides the full path back to an object from the root of roots.
- int32_t hpss_GetGID (gid_t ∗GID)

  Get the group id for the calling thread.
- hpss_read_queue_list_t hpss_GetIntersectedList (hpss_read_queue_list_t ∗List, hpss_read_queue_list_t ∗Partial)

  Return a list whose content is all items in the list which do appear in the sublist.
- int hpss_GetJunctionAttrs (const char ∗Path, hpss_Attrs_t ∗AttrOut)

  Retrieve file statistics associated with directory listing operations including junctions.
- int hpss_GetJunctions (uint32_t SubsystemID, uint64_t OffsetIn, uint32_t Entries, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_junction_ent_t ∗JentPtr)

  Retrives the junctions that are managed by the Core Server.
- hpss_errno_state_t hpss_GetLastHPSSErrno (void)

  Retrieves the error state of the last HPSS error.
- int hpss_GetListAttrs (const char ∗Path, hpss_Attrs_t ∗AttrOut)

  Retrieve file statistics associated with directory listing operations.
- hpss_reqid_t hpss_GetNextIORequestID ()

  Retrieves the next request id that will be used for I/O.
- int hpss_GetObjFromHandle (const ns_ObjHandle_t ∗ObjHandle, object_id_hash_t ∗ObjHash)

  Retrieves an object from a handle.
- uint32_t hpss_GetObjType (const ns_ObjHandle_t ∗ObjHandle)

  Retrieves the object type given its handle.
- int hpss_GetPathHandle (const ns_ObjHandle_t ∗ObjHandle, ns_ObjHandle_t ∗FilesetRootHandle, char ∗Path)

  Retrieves the pathname and root fileset that correspond with an Objhandle.
- int hpss_GetPathHandleObjHash (const object_id_hash_t ∗ObjIdHash, uint32_t SubsysId, ns_ObjHandle_t ∗FilesetRootHandle, ns_ObjHandle_t ∗ObjHandle, char ∗Path)

  Retrieves fileset, object handle, and path information using an object and subsystem.
- int hpss_GetPathObjHash (const object_id_hash_t ∗ObjIdHash, uint32_t SubsysId, ns_ObjHandle_t ∗FilesetRootHandle, char ∗Path)

  Retrieves the pathname and root fileset that correspond to an Object and Subsystem.
- int hpss_GetRawAttrHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗HandleOut, hpss_vattr_t ∗AttrOut)

  Obtains information about a symlink or junction using a handle.
- int hpss_GetSubSysLimits (uint32_t SubsystemID, const sec_cred_t ∗Ucred, hpss_subsys_limits_t ∗LimitsOut)

Retrieve subsystem limits.
- int hpss_GetSubSysStats (uint32_t SubsystemID, subsys_stats_t ∗StatsOut)

  Retrieve subsystem statistics.
- int hpss_GetThreadUcred (sec_cred_t ∗RetUcred)

  Retrieves the user credentials for the current thread.
- int hpss_GetTrashcan (const char ∗Path, char ∗TrashcanBuf, int TrashcanBufLength, ns_ObjHandle_-t ∗FilesetRoot)

  Locate the trashcan for a provided path.
- int hpss_GetTrashcanHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_-t ∗Ucred, char ∗TrashcanBuf, int TrashcanBufLength, ns_ObjHandle_t ∗FilesetRoot)

  Locate the trashcan for a provided handle/path.
- int hpss_GetTrashSettings (ns_TrashcanSettings_t ∗TrashSettings)

  Query the Root Core Server for trashcan settings.
- int32_t hpss_GetUID (uid_t ∗UID)

  Get the user id for the calling thread.
- void hpss_HashFlagsToString (unsigned16 flags, char ∗buf, int len)

  Returns a string representation of the provided hash flags.
- int hpss_InsertDistFile (const api_dist_file_info_t ∗FileInfo)

  Inserts a file table entry into the current file table.
- int hpss_JunctionCreate (const char ∗Path, const ns_ObjHandle_t ∗SourceHandle, ns_ObjHandle_t ∗JunctionHandle)

  Creates an HPSS junction.
- int hpss_JunctionCreateHandle (const ns_ObjHandle_t ∗ParentHandle, const char ∗Path, const ns_Obj-Handle_t ∗SourceHandle, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗JunctionHandle)

  Creates an HPSS junction in relation to a fileset handle.
- int hpss_JunctionDelete (const char ∗Path)

  Deletes a junction.
- int hpss_JunctionDeleteHandle (const ns_ObjHandle_t ∗ParentHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Deletes a junction.
- int hpss_Link (const char ∗Existing, const char ∗New)

  Creates a link to a file.
- int hpss_LinkHandle (const ns_ObjHandle_t ∗ObjHandle, const ns_ObjHandle_t ∗DirHandle, const char ∗New, const sec_cred_t ∗Ucred)

  Creates a link.
- int hpss_LinkHandleParent (const ns_ObjHandle_t ∗SrcDirHandle, const char ∗SrcFile, const ns_ObjHandle-_t ∗DestDirHandle, const char ∗DestFile, const sec_cred_t ∗Ucred)

  Creates a link.
- int hpss_LoadDefaultThreadState (uid_t UserID, mode_t Umask, const char ∗ClientFullName)

  Allows some clients to manipulate the global state of a thread.
- int hpss_LoadThreadState (uid_t UserID, mode_t Umask, const char ∗ClientFullName)

  Allows some clients to manipulate the local state of a thread.
- int hpss_LookupRootCS (const char ∗SiteName, hpss_srvr_id_t ∗ServerId)

  Retrieves the ID for a site's root Core Server.
- int64_t hpss_Lseek (int Fildes, int64_t Offset, int Whence)

  Sets the file offset for the open file handle.
- int hpss_Lstat (const char ∗Path, hpss_stat_t ∗Buf)

  Retrieve file, or link, statistics.
- int hpss_Migrate (int Fildes, uint32_t SrcLevel, uint32_t Flags, uint64_t ∗RetBytesMigrated)

  Migrates data in an open file from a specified hierarchy level.
- int hpss_MiscAdmin (uint32_t Function, char ∗TextP, ns_AdminConfArray_t ∗ValueConfArrayP)

  Accesses the Core Server's miscellaneous functions.

- int hpss_Mkdir (const char ∗Path, mode_t Mode)

    Creates a new directory.
- int hpss_MkdirHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, mode_t Mode, const sec_cred_t ∗Ucred, ns_ObjHandle_t ∗HandleOut, hpss_vattr_t ∗AttrOut)

    Creates a new directory.
- int hpss_Open (const char ∗Path, int Oflag, mode_t Mode, const hpss_cos_hints_t ∗HintsIn, const hpss_cos-_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

    Optionally create and open an HPSS file.
- int hpss_OpenBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, int Oflag, const sec_cred_t ∗Ucred)

    Open an HPSS file by bitfile id.
- int hpss_OpenBitfileVAttrs (const hpss_vattr_t ∗FileAttrs, int Oflag, const sec_cred_t ∗Ucred, hpss_cos_-hints_t ∗HintsOut, uint64_t ∗SegmentSize)

    Open a bitfile using vattrs.
- int hpss_Opendir (const char ∗DirName)

    Opens a directory stream.
- int hpss_OpendirHandle (const ns_ObjHandle_t ∗DirHandle, const sec_cred_t ∗Ucred)

    Open a directory via a handle.
- int hpss_OpenHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, int Oflag, mode_t Mode, sec-_cred_t ∗Ucred, const hpss_cos_hints_t ∗HintsIn, const hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut, hpss_vattr_t ∗AttrsOut)

    Optionally create and open an HPSS file relative to a given directory.
- int hpss_PingCore (const hpss_srvr_id_t ∗CoreServerID, uint32_t ∗RecvSecs, uint32_t ∗RecvUSecs)

    Pings the Core Server.
- int hpss_PIOEnd (hpss_pio_grp_t StripeGroup)

    End and clean up a parallel IO group context.
- int hpss_PIOExecute (int Fd, uint64_t FileOffset, uint64_t Size, const hpss_pio_grp_t StripeGroup, hpss_pio-_gapinfo_t ∗GapInfo, uint64_t ∗BytesMoved)

    Begin a PIO data transfer.
- int hpss_PIOExecuteItem (int Fd, uint64_t FileOffset, uint64_t Size, const hpss_pio_grp_t StripeGroup, hpss-_pio_gapinfo_t ∗GapInfo, uint64_t ∗BytesMoved, hpss_read_queue_item_t ∗Item)

    Begin a PIO data transfer, with a reference to a running read queue item.
- int hpss_PIOExportGrp (const hpss_pio_grp_t StripeGroup, void ∗∗Buffer, unsigned int ∗BufLength)

    Export a parallel IO group context to a buffer.
- int hpss_PIOFinalize (hpss_pio_grp_t ∗StripeGroup)

    Safely end and clean up a parallel IO group context.
- int hpss_PIOGetPartError (const hpss_pio_grp_t StripeGroup, int ∗Error)

    Retrieves a participant error code.
- int hpss_PIOGetRequestID (hpss_pio_grp_t StripeGroup, hpss_reqid_t ∗RequestID)

    Retrieves the request ID for a group.
- int hpss_PIOImportGrp (const void ∗Buffer, unsigned int BufLength, hpss_pio_grp_t ∗StripeGroup)

    Import a group which was exported with hpss_PIOExportGrp.
- int hpss_PIORegister (uint32_t StripeElement, const hpss_sockaddr_t ∗DataNetSockAddr, void ∗DataBuffer, uint32_t DataBufLen, hpss_pio_grp_t StripeGroup, const hpss_pio_cb_t IOCallback, const void ∗IOCallback-Arg)

    Register a PIO stripe participant.
- int hpss_PIOStart (hpss_pio_params_t ∗InputParams, hpss_pio_grp_t ∗StripeGroup)

    Start a new parallel I/O group context.
- signed32 hpss_PreferredReadOrder (uint32_t SubsysId, hpss_uuid_t ∗Context, sec_cred_t ∗UserCredsP, hpss_read_queue_list_t ∗ItemsToCheck, int Flags, int Delay, hpss_read_queue_list_t ∗ItemsReady, hpss_-read_queue_list_t ∗ItemsError)

    Request to be notified about ready items in a list of reads.

- int hpss_Purge (int Fildes, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, uint32_t Flags, uint64_t ∗RetBytesPurged)

  Purges data in an open file from a specified hierarchy level.
- int hpss_PurgeLock (int Fildes, purgelock_flag_t Flag)

  Locks or unlocks a file's purge state.
- void hpss_PurgeLoginCred (void)

  Purges an HPSS login credential.
- int hpss_PurgeOnMigrate (int Fildes, purgeonmigrate_flag_t Flag)

  Sets or clears the purge on migrate flag.
- ssize_t hpss_Read (int Fildes, void ∗Buf, size_t Nbyte)

  Reads a number of bytes from a file.
- int hpss_ReadAttrs (int Dirdes, uint64_t OffsetIn, uint32_t BufferSize, uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entries and object attributes from an open directory stream.
- int hpss_ReadAttrsHandle (const ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_t ∗Ucred, uint32_t BufferSize, uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entry attributes using a handle.
- int hpss_ReadAttrsPlus (int Dirdes, uint64_t OffsetIn, uint32_t BufferSize, hpss_readdir_flags_t Flags, uint32-_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read directory entries and object attributes from an open directory stream.
- int hpss_Readdir (int Dirdes, hpss_dirent_t ∗DirentPtr)

  Read directory entries from an open directory stream.
- int hpss_ReaddirHandle (const ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_t ∗Ucred, uint32_t BufferSize, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

  Read directory entries using a handle.
- int hpss_ReaddirPlus (int Dirdes, uint64_t OffsetIn, uint32_t BufferSize, hpss_readdir_flags_t Flags, uint32_t ∗End, uint64_t ∗OffsetOut, hpss_dirent_t ∗DirentPtr)

  Read directory entries using a handle.
- ssize_t hpss_ReadItem (int Fildes, void ∗Buf, size_t Nbyte, hpss_read_queue_item_t ∗Item)

  Reads a number of bytes from a file using a read queue item.
- int hpss_Readlink (const char ∗Path, char ∗Contents, size_t BufferSize)

  Retrieves the contents of a symbolic link.
- int hpss_ReadlinkHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, char ∗Contents, size_t BufferSize, const sec_cred_t ∗Ucred)

  Retrieves the contents of a symbolic link.
- int hpss_ReadList (const hpss_IOD_t ∗IODPtr, uint32_t Flags, hpss_IOR_t ∗IORPtr)

  Send a read IOD to HPSS.
- signed32 hpss_ReadQueueAdd (uint32_t SubsysId, hpss_uuid_t ∗Context, sec_cred_t ∗UserCredsP, hpss-_read_queue_item_t ∗Item)

  Add an item from the provided read queue.
- signed32 hpss_ReadQueueAddMany (uint32_t SubsysId, hpss_uuid_t ∗Context, sec_cred_t ∗UserCredsP, hpss_read_queue_list_t ∗ItemList)

  Add items from the provided read queue.
- signed32 hpss_ReadQueueCreateContext (uint32_t SubsysId, sec_cred_t ∗UserCredsP, hpss_uuid_t ∗Context)

  Create a new read queue context.
- signed32 hpss_ReadQueueList (uint32_t SubsysId, hpss_uuid_t ∗Context, sec_cred_t ∗UserCredsP, hpssoid_t ∗VVID, unsigned32 Start, unsigned32 Max, hpss_read_queue_list_t ∗List)

  List items in the provided read queue.
- signed32 hpss_ReadQueueRemove (uint32_t SubsysId, hpss_uuid_t ∗Context, sec_cred_t ∗UserCredsP, hpss_reqid_list_t ∗RequestList)

  Remove items from the provided read queue.

- signed32 hpss_ReadQueueRemoveContext (uint32_t SubsysId, hpss_uuid_t ∗ContextP, sec_cred_t ∗User-CredsP, int Flags)

  Remove a read queue context.
- int hpss_ReadRawAttrsHandle (const ns_ObjHandle_t ∗ObjHandle, uint64_t OffsetIn, const sec_cred_-t ∗Ucred, uint32_t BufferSize, uint32_t GetAttributes, uint32_t ∗End, uint64_t ∗OffsetOut, ns_DirEntry_t ∗DirentPtr)

  Read raw attributes using a handle; does not cross junctions.
- int hpss_ReadTrash (int32_t SubsystemId, const uint32_t ∗UserIdP, const uint32_t ∗RealmIdP, uint32_-t BufferSize, uint32_t ∗End, ns_TrashEntry_t ∗TrashPtr)

  Read trash entries for a subsystem.
- hpss_read_queue_list_t hpss_RemoveIntersectionFromList (hpss_read_queue_list_t ∗List, hpss_read_-queue_list_t ∗Partial)

  Return a list whose content is all items in the list which do not appear in the sublist.
- int hpss_Rename (const char ∗Old, const char ∗New)

  Rename an HPSS file or directory.
- int hpss_RenameHandle (const ns_ObjHandle_t ∗OldHandle, const char ∗OldPath, const ns_ObjHandle_t ∗NewHandle, const char ∗NewPath, const sec_cred_t ∗Ucred)

  Rename an HPSS file or directory using an object handle.
- int hpss_ReopenBitfile (int Fildes, const bfs_bitfile_obj_handle_t ∗BitfileObj, int Oflag, const sec_cred_t ∗Ucred)

  Open an HPSS file by bitfile object handle.
- int hpss_ResetSubSysStats (uint32_t SubsystemID, subsys_stats_t ∗StatsOut)

  Reset subsystem statistics.
- int hpss_Rewinddir (int Dirdes)

  Rewind a directory stream back to the beginning.
- int hpss_Rmdir (const char ∗Path)

  Remove a directory.
- int hpss_RmdirHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Remove a directory using a handle.
- int hpss_RmdirHandleImmediate (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Remove a directory using a handle, bypassing the trashcan.
- int hpss_RmdirImmediate (const char ∗Path)

  Remove a directory, bypassing the trashcan.
- int hpss_Rumble (int32_t SubsystemId, timespec_t ∗TimeP, uint32_t BufferSize, uint32_t ∗End, ns_Rumble-Entry_t ∗RumblePtr)

  Read rumble entries for a subsystem.
- int hpss_SetAcct (acct_rec_t NewCurAcct)

  Sets the current account code.
- int hpss_SetAcctByName (const char ∗NewAcctName)

  Sets the current account name.
- int hpss_SetACL (const char ∗Path, uint32_t Options, const ns_ACLConfArray_t ∗ACL)

  Sets the ACL entries of a file or directory.
- int hpss_SetACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t Options, const ns_ACLConfArray_t ∗ACL)

  Sets the ACL entries of a file or directory using a handle.
- int hpss_SetAPILogLevel (int LogLevel)

  Modifies the Client API log level.
- int hpss_SetAPILogPath (const char ∗LogPath)

  Modifies the Client API log path.
- void hpss_SetApplicationName (const char ∗App)

  Modifies the Client API application name.

- int hpss_SetAppLoginCred (const char ∗App, const char ∗PrincipalName, hpss_authn_mech_t Mechanism, hpss_rpc_cred_type_t CredType, hpss_rpc_auth_type_t AuthType, const void ∗Authenticator)

  Sets an HPSS login credential.
- int hpss_SetAttrHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint64_t SelFlagsIn, const hpss_vattr_t ∗AttrIn, uint64_t ∗SelFlagsOut, hpss_vattr_t ∗AttrOut)

  Updates attribute values for a file or directory.
- int hpss_SetAuditInfo (hpss_sockaddr_t EndUserHostAddr)

  Sets audit information for threads.
- int hpss_SetConfiguration (const api_config_t ∗ConfigIn)

  Sets the values used for the API configuration settings.
- int hpss_SetCOSByHints (int Fildes, uint32_t Flags, const hpss_cos_hints_t ∗HintsPtr, const hpss_cos_-priorities_t ∗PrioPtr, hpss_cos_md_t ∗COSPtr)

  Sets the COS of an empty file.
- int hpss_SetFileNotrunc (const char ∗Path, notrunc_flag_t Flag)

  Sets or clears a file's NOTRUNC_FINAL_SEG flag.
- int hpss_SetFileOffset (int Fildes, uint64_t OffsetIn, int Whence, int Direction, uint64_t ∗OffsetOut)

  Sets the file offset for an open file handle.
- int hpss_SetGID (uint32_t NewCurGid)

  Set the group id for the calling thread.
- int hpss_SetLoginCred (const char ∗PrincipalName, hpss_authn_mech_t Mechanism, hpss_rpc_cred_type_t CredType, hpss_rpc_auth_type_t AuthType, const void ∗Authenticator)

  Sets an HPSS login credential.
- int32_t hpss_SetNextIORequestID (hpss_reqid_t ∗RequestId)

  Set the request ID to be used for the next I/O in this thread. This should be a request ID generated from hpss_Get-UniqueRequestID or from hpss_ReadQueueAdd.
- int hpss_SiteIdToName (const hpss_id_t ∗SiteId, char ∗SiteName)

  Convert Site ID to Site Name.
- int hpss_SiteNameToId (const char ∗SiteName, hpss_id_t ∗SiteId)

  Convert Site Name to Site ID.
- int hpss_Stage (int Fildes, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, uint32_t Flags)

  Stage an open HPSS file.
- int hpss_StageBatch (hpss_stage_batch_t ∗Batch, hpss_stage_batch_status_t ∗Status)

  Stage a batch of bitfiles.
- int hpss_StageBatchCallBack (hpss_stage_batch_t ∗Batch, bfs_callback_addr_t ∗CallBackPtr, hpss_reqid_t ∗ReqID, hpss_stage_bitfile_list_t ∗BFIDs, hpss_stage_batch_status_t ∗Status)

  Stage a batch of bitfiles asynchronously.
- int hpss_StageCallBack (const char ∗Path, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, bfs_-callback_addr_t ∗CallBackPtr, uint32_t Flags, hpss_reqid_t ∗ReqID, bfs_bitfile_obj_handle_t ∗BitfileObj)

  Stage an HPSS file in the background.
- int hpss_StageCallBackBitfile (const bfs_bitfile_obj_handle_t ∗BitfileObj, uint64_t Offset, uint64_t Length, uint32_t StorageLevel, bfs_callback_addr_t ∗CallBackPtr, uint32_t Flags, hpss_reqid_t ∗ReqID)

  Stage a file in the background using its bitfile id.
- int hpss_Stat (const char ∗Path, hpss_stat_t ∗Buf)

  Retrieve file statistics.
- int hpss_Statfs (uint32_t CosId, hpss_statfs_t ∗StatfsBuffer)

  Retrieve file system status information for a class of service.
- int hpss_Statfs64 (uint32_t CosId, hpss_statfs64_t ∗StatfsBuffer)

  Retrieve file system status information for a class of service.
- int hpss_Statvfs (uint32_t CosId, hpss_statvfs_t ∗StatvfsBuffer)

  Retrieve file system status information for a class of service in VFS format.
- int hpss_Statvfs64 (uint32_t CosId, hpss_statvfs64_t ∗StatvfsBuffer)

  Retrieve file system status information for a class of service in VFS format.

- int hpss_Symlink (const char ∗Contents, const char ∗Path)

  Create a symbolic link.
- int hpss_SymlinkHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Contents, const char ∗Path, const sec_cred_t ∗Ucred, hpss_vattr_t ∗AttrsOut)

  Create a symbolic link using a handle.
- int hpss_ThreadCleanUp (void)

  Cleans up a thread's context.
- int hpss_Truncate (const char ∗Path, uint64_t Length)

  Truncate a file.
- int hpss_TruncateHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, uint64_t Length, const sec-_cred_t ∗Ucred)

  Truncate a file using a handle.
- mode_t hpss_Umask (mode_t CMask)

  Set the file mode creation mask, and returns the previous mask value.
- int hpss_Undelete (const char ∗Path, hpss_undelete_flags_t Flags)

  Restore a namespace entry from the trash.
- int hpss_UndeleteHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, hpss_undelete_flags_t Flags)

  Restore a namespace entry from the trash.
- int hpss_Unlink (const char ∗Path)

  Remove a namespace entry.
- int hpss_UnlinkHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Remove a namespace entry using a handle.
- int hpss_UnlinkHandleImmediate (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred)

  Remove a namespace entry using a handle, bypassing the trashcan.
- int hpss_UnlinkImmediate (const char ∗Path)

  Remove a namespace entry, bypassing the trashcan.
- int hpss_UpdateACL (const char ∗Path, int32_t Options, const ns_ACLConfArray_t ∗ACL)

  Updates an ACL array of a file or directory.
- int hpss_UpdateACLHandle (const ns_ObjHandle_t ∗ObjHandle, const char ∗Path, const sec_cred_t ∗Ucred, uint32_t Options, const ns_ACLConfArray_t ∗ACL)

  Updates an ACL entry of a file or directory using a handle.
- int hpss_UserAttrCloseCursor (int SubsystemId, const hpss_cursor_id_t ∗CursorId)

  Close a cursor.
- int hpss_UserAttrDeleteAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, const hpss_userattr_list_t ∗Attr, const char ∗Schema)

  Delete attributes from a namespace path using a handle.
- int hpss_UserAttrDeleteAttrs (const char ∗Path, const hpss_userattr_list_t ∗Attr, const char ∗Schema)

  Delete User-defined Attributes on a namespace path.
- int hpss_UserAttrGetAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, hpss_userattr_list_t ∗Attr, int XMLFlag, int XMLSize)

  Retrieve User-defined Attributes on a namespace path using a handle.
- int hpss_UserAttrGetAttrs (const char ∗Path, hpss_userattr_list_t ∗Attr, int XMLFlag, int XMLSize)

  Retrieve User-defined Attributes on a namespace path.
- int hpss_UserAttrListAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, hpss_userattr_list_t ∗Attr, int Flags, int XMLSize)

  List User-defined Attributes associated with a namespace path using a handle.
- int hpss_UserAttrListAttrs (const char ∗Path, hpss_userattr_list_t ∗Attr, int Flags, int XMLSize)

  List User-defined Attributes associated with a namespace path.
- int hpss_UserAttrReadObj (int SubsystemId, object_id_hash_list_t ∗ObjectList, hpss_cursor_id_t ∗CursorId)

  Read objects from a previous object search.

- int hpss_UserAttrReadXML (int SubsystemId, hpss_string_list_t ∗XML, hpss_cursor_id_t ∗CursorId)

  Read object ids from a previous XML search.
- int hpss_UserAttrReadXMLObj (int SubsystemId, object_id_hash_list_t ∗ObjectList, hpss_string_list_t ∗XML, hpss_cursor_id_t ∗CursorId)

  Read more XML/objects from a previous XML/Object search.
- int hpss_UserAttrSetAttrHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, const hpss_userattr_list_t ∗Attr, const char ∗Schema)

  Set User-defined Attributes on a namespace path using a handle.
- int hpss_UserAttrSetAttrs (const char ∗Path, const hpss_userattr_list_t ∗Attr, const char ∗Schema)

  Set User-defined Attributes on a namespace path.
- int hpss_UserAttrXQueryGet (const char ∗Path, const char ∗XQuery, char ∗Buffer, int BufSize)

  Retrieve User-defined Attribute data for a path using XQuery.
- int hpss_UserAttrXQueryGetHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_-t ∗Ucred, const char ∗XQuery, char ∗Buffer, int XMLSize)

  Retrieve User-defined Attribute data for a path using XQuery and an object handle.
- int hpss_UserAttrXQueryUpdate (const char ∗Path, const char ∗XQuery, const char ∗Schema)

  Update User-defined Attributes on a path using an XQuery string.
- int hpss_UserAttrXQueryUpdateHandle (const ns_ObjHandle_t ∗Obj, const char ∗Path, const sec_cred_t ∗Ucred, const char ∗XQuery, const char ∗Schema)

  Update User-defined Attributes on a path using an XQuery string and a handle.
- int hpss_Utime (const char ∗Path, const struct utimbuf ∗Times)

  Set the access and modification times of a file.
- ssize_t hpss_Write (int Fildes, const void ∗Buf, size_t Nbyte)

  Write data to an open file.
- int hpss_WriteList (const hpss_IOD_t ∗IODPtr, uint32_t Flags, hpss_IOR_t ∗IORPtr)

  Issue a write IOD to HPSS.

11.68.1 Class Documentation

11.68.1.1 struct api_dist_file_info_t::file_info

Information about the file entry.

Class Members

hpss_object_-handle_t	bitfile_handle	HPSS file bitfile handle
hpss_srvr_id_t	CoreServerID	Core Server UUID
uint32_t	FilesetCOS	COS assigned to the file
uint64_t	Offset	File offset
int	OpenFlag	Specifies the file status and access modes. Specifies the file status and access modes to be assigned. Applicable values may be bitwise OR'd together. Refer to POSIX.1 for specific be-havior. O_RDONLY O_WRONLY O_RDWR O_APPEND O_CREAT O_EXCL O_TRUNC

11.68.1.2 struct hpss_file_hash_digest

Defined the input/output structure for hpss_FgetFileDigest and hpss_FsetFileDigest and hpss_FgetFileDigestList and hpss_FsetFileDigestList.

Class Members

char	Buffer[HPSS_M-AX_HASH_DIG-EST_BYTES]	Hash digest
char	Creator[HPSS_-MAX_FILE_HA-SH_CREATOR-_NAME]	Creator supplied text
hpss_file_hash_-flags_t	Flags	Hash state and control flags
timestamp_sec-_t	ModifyTime	Last modification time (read only)
hpss_hash_-type_t	Type	Type of specified hash

11.68.1.3 struct hpss_file_hash_digest_list

A list of hash digests for single or multi stripe digest handling.

For multi stripe, items in the digest list should are expected to be in their correct stripe order; this is the order they will be retrieved in.

Class Members

struct hpss_file_-hash_digest_list	List	List container

11.68.2 Typedef Documentation

11.68.2.1 typedef enum **hpss_file_hash_flags hpss_file_hash_flags_t**

Defines the file hash flag values.

- Read-only flags:
  - HPSS_FILE_HASH_SET_BY_USER
  - HPSS_FILE_HASH_MIGRATE_VERIFIED
- Read/update flags:
  - HPSS_FILE_HASH_DIGEST_VALID
  - HPSS_FILE_HASH_SKIP_VERIFICATION
  - HPSS_FILE_HASH_GENERATED
  - HPSS_FILE_HASH_DIGEST_ACTIVE (valid for multi-hash only)

11.68.3 Enumeration Type Documentation

11.68.3.1 enum **hpss_file_hash_flags**

Defines the file hash flag values.

- Read-only flags:

  - HPSS_FILE_HASH_SET_BY_USER
  - HPSS_FILE_HASH_MIGRATE_VERIFIED
- Read/update flags:
  - HPSS_FILE_HASH_DIGEST_VALID
  - HPSS_FILE_HASH_SKIP_VERIFICATION
  - HPSS_FILE_HASH_GENERATED
  - HPSS_FILE_HASH_DIGEST_ACTIVE (valid for multi-hash only)

Enumerator

HPSS_FILE_HASH_DIGEST_VALID Hash digest buffer is valid

HPSS_FILE_HASH_SET_BY_USER Hash provided by user

HPSS_FILE_HASH_MIGRATE_VERIFIED Hash verified against data on migration

HPSS_FILE_HASH_SKIP_VERIFICATION Skip hash verification on migration

HPSS_FILE_HASH_GENERATED Hash was generated by app

HPSS_FILE_HASH_DIGEST_ACTIVE Hash is active and can be used - this value is always set for single stripe hashes, but can vary for multi hash

11.68.3.2 enum hpss_file_hash_stripe_flags

Preferencing flags for digest list operations.

Enumerator

HPSS_HASH_SINGLE_ONLY Only operate on single stripe

HPSS_HASH_SINGLE Prefer single stripe

HPSS_HASH_STRIPE Prefer multi stripe

HPSS_HASH_STRIPE_ONLY Only operate on multi stripe

11.68.3.3 enum hpss_readdir_flags_t

Flags for hpss_ReadAttrPlus and hpss_ReaddirPlus

Enumerator

HPSS_READDIR_NONE No flags

HPSS_READDIR_GETATTRS Get attributes

HPSS_READDIR_RAW Don’t chase junctions

HPSS_READDIR_NFS Retrieve in objid sorted order

11.68.4 Function Documentation

11.68.4.1 void API_FreeIOR (hpss_IOR_t ∗ *IORPtr*)

Frees memory that was allocated to an IOR.

This routine frees memory that was allocated for a client IOR.

Parameters

in,out	*IORPtr*	pointer to IOR to free

11.68.4.2 int API_GetBatchStatus (hpss_stage_batch_status_t ∗ *Status,* int ∗ *Rc*)

Retrieve Batch Status.

Traverses a batch status structure and determines if any errors occurred.

Parameters

in	*Status*	Batch Status Structure
in,out	*Rc*	Batch Status Indicator

Return values

-EFAULT	NULL Status Structure
-EFAULT	NULL Return Code
-EINVAL	Invalid Status List Length
-EINVAL	NULL Status List Structure
0	Success

Note

The status of the first failure is returned, but there could be other failures subsequent to that one.

11.68.4.3 int API_GetClientAddr (const iod_srcsinkdesc_t ∗ *SrcSinkPtr,* const iod_srcsinkreply_t ∗ *SrcSinkReplyPtr,* uint64_t *Offset,* iod_address_t ∗∗ *RetAddr,* uint64_t ∗ *RetLength,* iod_srcsinkdesc_t ∗∗ *RetDescPtr,* iod_srcsinkreply_t ∗∗ *RetReplyPtr*)

Retrieves the client address for a data transfer.

This routine determines the client address which is associated with the current offset within the transfer, as well as the number of subsequent bytes which also correspond to that address. Also returned are pointers to the source/sink descriptor and reply structures which correspond the returned address.

Parameters

in	*SrcSinkPtr*	initial src/sink descriptor
in	*SrcSinkReplyPtr*	initial src/sink reply
in	*Offset*	offset within transfer
out	*RetAddr*	returned client length
out	*RetLength*	returned length
out	*RetDescPtr*	returned descriptor ptr
out	*RetReplyPtr*	returned reply ptr

Return values

0	Success
-EFAULT	Could not find client address

11.68.4.4 void API_StageBatchFree (hpss_stage_batch_t ∗ *Batch*)

Free a Batch Stage Request Structure.

Free a Batch Stage Request Structure

Parameters

`in,out`	*Batch*	Batch Request Structure

11.68.4.5 int API_StageBatchInit (**hpss_stage_batch_t** ∗ *Batch,* int *len*)

Initialize a Batch Stage Request Structure.

Create a Batch Stage Request Structure of the specified length

Parameters

`in,out`	*Batch*	Batch Request Structure
`in`	*len*	Batch Request Length

Return values

HPSS_E_NOERROR	Success
HPSS_EFAULT	NULL Batch Structure Provided
HPSS_ERANGE	The size of the batch is larger than the legal size. Split the batch into multiple batches of HPSS_MAX_STAGE_BATCH_LEN or smaller and try again.
HPSS_ENOMEM	Could not allocate batch structure

Note

The list structure is initialized with all zero values and the correct list length

11.68.4.6 void API_StageStatusFree (**hpss_stage_batch_status_t** ∗ *Status*)

Free a Batch Stage Status Structure.

Free a Batch Stage Status Structure

Parameters

`in,out`	*Status*	Batch Status Structure

11.68.4.7 **hpss_read_queue_list_t** hpss_GetIntersectedList (**hpss_read_queue_list_t** ∗ *List,* **hpss_read_queue_list_t** ∗ *Partial*)

Return a list whose content is all items in the list which do appear in the sublist.

Parameters

`in`	*List*	List of all items to process
`in`	*Partial*	Sublist of items to include

Returns

A new allocation of list items

Note

The new list array must be freed

11.68.4.8 int32_t hpss_GetUID (uid_t ∗ *UID*)

Get the user id for the calling thread.

client_api

The 'hpss_GetUID' function gets the current UID for the calling thread.

Return values

0	Success
<0	Error from API_ClientAPIInit().

11.68.4.9 int hpss_Readlink (const char ∗ *Path,* char ∗ *Contents,* size_t *BufferSize*)

Retrieves the contents of a symbolic link.

The 'hpss_Readlink' function returns the contents of the named symbolic link.

Parameters

`in`	*Path*	Name of the link
`out`	*Contents*	Contents of the link
`in`	*BufferSize*	Size, in bytes, of Contents

Return values

0	Contents contains symlink data.
-EACCES	Search permission is denied on a component of the path prefix, or read permission is denied on the symbolic link.
-EFAULT	The Path or Contents parameter is a NULL pointer.
-EINVAL	The specified file is not a symbolic link.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The specified path name does not exist.
-ENOTDIR	A component of the Path prefix is not a directory.
-ERANGE	The size of the Contents buffer is not big enough to contain the contents of the symbolic link or the value of the BufferSize parameter is zero.

11.68.4.10 int hpss_ReadlinkHandle (const ns_ObjHandle_t ∗ *ObjHandle,* const char ∗ *Path,* char ∗ *Contents,* size_t *BufferSize,* const sec_cred_t ∗ *Ucred*)

Retrieves the contents of a symbolic link.

The 'hpss_ReadlinkHandle' function returns the contents of the symbolic link with the name 'Path' (taken relative to 'ObjHandle').

Parameters

`in`	*ObjHandle*	handle of parent
`in`	*Path*	Name of symlink
`in`	*BufferSize*	Size, in bytes, of Contents
`out`	*Contents*	Contents of the link

in	*Ucred*	user credentials

Return values

0	'Contents' contains the symlink data.
-EACCES	Search permission is denied on a component of the path prefix, or read permission is denied on the symbolic link.
-EFAULT	The ObjHandle, Path or Contents parameter is a NULL pointer.
-EINVAL	The specified file is not a symbolic link.
-ENAMETOOLONG	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
-ENOENT	The specified path name does not exist.
-ENOTDIR	A component of the Path prefix is not a directory.
-ERANGE	The size of the Contents buffer is not big enough to contain the contents of the symbolic link or the value of the BufferSize parameter is zero.

11.68.4.11 int hpss_ReadList (const hpss_IOD_t ∗ *IODPtr,* uint32_t *Flags,* hpss_IOR_t ∗ *IORPtr*)

Send a read IOD to HPSS.

Parameters

in	*IODPtr*	IO description
in	*Flags*	IO specific flags
out	*IORPtr*	results of the IO

The 'hpss_ReadList' function reads the file data specified by the source descriptor list in the IOD pointed to by 'IODPtr' and moves the data to destinations specified by the sink descriptor list in the IOD. Results of the request will be returned in the structure pointed to by 'IORPtr'.

Return values

0	- No error.

11.68.4.12 hpss_read_queue_list_t hpss_RemoveIntersectionFromList (hpss_read_queue_list_t ∗ *List,* hpss_read_queue_list_t ∗ *Partial*)

Return a list whose content is all items in the list which do not appear in the sublist.

Parameters

in	*List*	List of all items to process
in	*Partial*	Sublist of items to remove

Returns

A new allocation of list items

Note

The new list array must be freed

11.68.4.13 int hpss_Rumble (signed32 *SubsystemId,* timespec_t ∗ *TimeP,* unsigned32 *BufferSize,* unsigned32 ∗ *End,* ns_RumbleEntry_t ∗ *RumblePtr*)

Read rumble entries for a subsystem.

The 'hpss_Rumble' function returns an array of rumble entries, RumblePtr, representing changes to the HPSS namespace since TimeP for subsystem SubsystemId. The number of entries is limited by BufferSize, which describes the size of the allocated RumblePtr structure.

Parameters

`in`	*SubsystemId*	the subsystem for the core server
`in`	*TimeP*	return changes after time specified
`in`	*BufferSize*	size of output buffer (in bytes)
`out`	*End*	hit end of rumble entries
`out`	*RumblePtr*	rumble entry information

Return values

>=0	No error. Return value is the number of entries copied to the output buffer.
-ERANGE	The buffer size was not large enough to return entries
-EFAULT	One of the End or RumblePtr parameters is NULL.
-EINVAL	BufferSize is zero.

Note

rumble paths are from their fileset root. For systems which make use of filesets this may not be an absolute path. To obtain an absolute path, the handle should be traversed back to the root of roots. See hpss_GetFullPath() for more information.

11.68.4.14 int hpss_WriteList (const hpss_IOD_t * *IODPtr,* uint32_t *Flags,* hpss_IOR_t * *IORPtr*)

Issue a write IOD to HPSS.

Parameters

`in`	*IODPtr*	IO description
`in`	*Flags*	IO specific flags
`out`	*IORPtr*	results of the IO

The 'hpss_WriteList' function writes the file data specified by the source descriptor list in the IOD pointed to by 'IODPtr' and moves the data to destinations specified by the sink descriptor list in the IOD. Results of the request will be returned in the structure pointed to by 'IORPtr'.

Return values

0	Success

Note

hpss_WriteList should not be directly used by client applications; it is meant to be used with internal HPSS client applications only.

11.69 hpss_api_def.x File Reference

Client API definitions used in RPC calls.

Classes

- struct hpss_fileattr

  HPSS file attributes and bits. More...

- struct hpss_xfileattr

  HPSS extended file attributes The HPSS extended file attributes include bitfile and storage server information on the file. It also contains information that must be freed by the caller - see functions which provide this structure for more info. More...

Typedefs

- typedef hpss_AttrBits_t hpss_fileattrbits_t

  Specifies the HPSS file attribute bits.

11.69.1 Detailed Description

Client API definitions used in RPC calls.

11.69.2 Class Documentation

11.69.2.1 struct hpss_fileattr

HPSS file attributes and bits.

Class Members

hpss_Attrs_t	Attrs	File Attributes
ns_ObjHandle_t	ObjectHandle	File Object Handle

11.69.2.2 struct hpss_xfileattr

HPSS extended file attributes The HPSS extended file attributes include bitfile and storage server information on the file. It also contains information that must be freed by the caller - see functions which provide this structure for more info.

Class Members

hpss_Attrs_t	Attrs	Standard File Attributes
ns_ObjHandle_t	ObjectHandle	File Object Handle
bf_sc_attrib_t	SCAttrib[HPSS-_MAX_STORA-GE_LEVELS]	Extended Attributes

11.70 hpss_cos.x File Reference

Defines class of service data definitions.

Classes

- struct hpss_cos_hints
- struct hpss_cos_md
- struct hpss_cos_priorities

  Class of Service Priorities. More...

11.70.1 Detailed Description

Defines class of service data definitions.

11.70.2 Class Documentation

11.70.2.1 struct hpss_cos_hints

Define structure for COS hints provided by HPSS client

Class Members

uint32_t	Access-Frequency	Access frequency
uint32_t	AvgLatency	Average lateny
uint32_t	COSId	COS ID
fstring	COSName[HPS-S_MAX_OBJE-CT_NAME]	COS Name
uint32_t	FamilyId	File family Id
uint32_t	Flags	Flags
uint64_t	MaxFileSize	Maximum file size
uint64_t	MinFileSize	Minimum file size
uint64_t	Optimum-AccessSize	Optimum Access Size
uint32_t	ReadOps	Read operations allowed
uint32_t	StageCode	Stage type
uint64_t	StripeLength	Stripe length
uint32_t	StripeWidth	Stripe width
uint32_t	TransferRate	Transfer rate
uint32_t	WriteOps	Write operations allowed

11.70.2.2 struct hpss_cos_md

Structure for class of service metadata entry

Class Members

uint32_t	Access-Frequency	Specifies the expected access frequency. Specifies the expected rate of access for the bitfile. • FREQ_HOURLY • FREQ_DAILY • FREQ_WEEKLY • FREQ_MONTHLY • FREQ_ARCHIVE

uint32_t	AllocMethod	Specifies the method for allocation of top level disk segments. The method to use for the allocation of disk storage segments at the top level of the hierarchy for this file. If the top level of the hierarchy is tape, this field is ignored. Supported methods are: • ALLOC_CLASSIC All segments are the same size, except the last segment which may be truncated. The segment size will be computed using the file size, maximum and minimum segment sizes, and the average number of segments per file. • ALLOC_MAXSIZE All segments will use the maximum segment size configured for this storage class, except the last segment which may be truncated. • ALLOC_VARSIZE The first segment will use the minimum segment size configured for this storage class. The size of each successive segment will be twice preceding segment until the maximum segment size is reached. All successive segments after the maximum is reached are the maximum segment size. The last segment may be truncated to best fit the remaining data.
uint32_t	AvgLatency	Average latency between request reception and start of data transfer. (generally only nonzero for COS where requests may result in use of mountable media
uint32_t	COSId	Class of service ID
fstring	COSName[HPSS_MAX_OBJECT_NAME]	Name for this Class of Service
hpss_hash_type_t	FileHashType	Specifies the algorithm to use for migration file hash. The algorithm to use when generating file hash digests during migrations to tape for files stored with this class of service. See hpss_types.x for information about valid file hash algorithm types.
uint32_t	Flags	Optional flags for bitfile behavior. Optionally specify any combination of the following options: • COS_ENFORCE_MAX_FILE_SIZE If ON, bitfiles with a size greater than MaxFileSize cannot be created in this COS. Attempts to do so will result in the request being rejected with an error. • COS_FORCE_SELECTION If ON, a client must explicitly select this COS in order to have a file assigned to it. If the client merely supplies general COS hints for a bitfile, this COS will not be selected. • COS_AUTO_STAGE_RETRY If ON, HPSS will automatically retry a failed stage from the primary copy if a valid secondary copy exists. • COS_AUTO_READ_RETRY If ON, and a valid secondary copy of the data exists, and an attempt to read the first copy fails, HPSS will automatically retry the read using the secondary copy. • COS_TRUNC_FINAL_SEG If ON, the final segment of the file will be truncated on file close to the smallest valid segment size which will hold the data. The user may turn off truncation for an individual file by specifying the HINTS_NOTRUNC_FINAL_SEG in his hints Flags on file creation or when setting the COS by hints for an empty file (hpss_SetCOSByHints or hpss_Fcntl). There is no mechanism to allow the user to turn truncation on if the COS does not allow it. • COS_FULL_AGGR_RECALL If ON, when a user requests the stage of a file which resides in a tape aggregate, HPSS will stage every file in the aggregate. • COS_AUTO_CHANGE_COS If ON, upon disk migration the file will be examined to see if it's in the correct COS (determined by Min/MaxFileSize). If the file is better suited for another COS, then the file will undergo a Change in COS before being migrated.

uint32_t	HierId	Id of hierarchy supporting this COS
uint64_t	MaxFileSize	Max size file(bytes) which can be stored in this COS
hpss_migr_-order_type_e	MigrOrder	Specifies the Migration Order.
uint64_t	MinFileSize	Min size file(bytes) which can be stored in this COS
uint32_t	Optimum-AccessSize	Optimum Access Size for reading files stored in this COS (bytes)
uint32_t	ReadOps	Specifies the valid read operations. Specifies the valid read operations for the bitfile: • HPSS_OP_READ Allow read operations.
uint32_t	StageCode	Specifies the desired staging behavior. Specifies the staging behavior desired: • COS_STAGE_NO_STAGE File is not to be staged on open. The data will be read from the current level in the hierarchy, or data may be explicitly staged by the client. • COS_STAGE_ON_OPEN Entire file is to be staged to the top level in the hierarchy before open returns. • COS_STAGE_ON_OPEN_ASYNC Entire file is to be staged to the top level in the hierarchy without blocking in open. Reads/writes are blocked only until the portion of the file being accessed is staged. • COS_STAGE_ON_OPEN_BACKGROUND File is to be staged in a background task.
uint32_t	TransferRate	Transfer rate (KB)
uint32_t	WriteOps	Specifies the valid write operations. Specifies the valid write operations for the bitfile: • HPSS_OP_WRITE Allow write operations • HPSS_OP_APPEND Allow append operations

11.70.2.3 struct hpss_cos_priorities

Class of Service Priorities.

Assists a client in selecting a CCOS for a bitfile; valid priority values are:

- NO_PRIORITY
- LOWEST_PRIORITY
- LOW_PRIORITY
- DESIRED_PRIORITY
- HIGHLY_DESIRED_PRIORITY
- REQUIRED_PRIORITY

Class Members

uint32_t	Access-Frequency-Priority	Priority of access frequency
uint32_t	AvgLatency-Priority	Priority of the latency
uint32_t	COSIdPriority	Priority of the COS Id
uint32_t	COSName-Priority	Priority of the COS Name
uint32_t	FamilyIdPriority	Priority of the file family
uint32_t	MaxFileSize-Priority	Priority of the max file size
uint32_t	MinFileSize-Priority	Priority of the min file size
uint32_t	Optimum-AccessSize-Priority	Priority of the access size
uint32_t	ReadOpsPriority	Priority of the read ops
uint32_t	StageCode-Priority	Priority of the stage code
uint32_t	StripeLength-Priority	Priority of the stripe length
uint32_t	StripeWidth-Priority	Priority of the stripe width
uint32_t	TransferRate-Priority	Priority of the transfer rate
uint32_t	WriteOpsPriority	Priority of the write ops

11.71 hpss_dirent.h File Reference

```
#include <sys/types.h>
#include "hpss_types.h"
#include "hpss_limits.h"
#include "ns_interface_defs.h"
```

Classes

- struct hpss_dirent

  HPSS Directory Entry. More...

Typedefs

- typedef struct hpss_dirent hpss_dirent_t

  HPSS Directory Entry.

11.71.1 Detailed Description

This file contains the definition for the hpss_dirent structure, which stores HPSS directory name entry information.

11.71.2 Class Documentation

11.71.2.1 struct hpss_dirent

HPSS Directory Entry.

Class Members

ns_ObjHandle_t	d_handle	HPSS Name Server handle
char	d_name[HPSS_-MAX_FILE_NA-ME]	Directory entry name
unsigned16	d_namelen	Length of entry name
u_signed64	d_offset	Offset of next entry
unsigned16	d_reclen	Record length

11.72 hpss_Getenv.c File Reference

Read and Update HPSS Environment variables.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h>
#include <pwd.h>
#include <sys/utsname.h>
#include <errno.h>
#include "hpss_thread_safe.h"
#include "hpss_limits.h"
#include "hpss_Getenv.h"
#include "hpss_env_defs.h"
#include "hpss_errno.h"
#include "hpss_StringUtil.h"
```

Functions

- signed32 hpss_Envcpy (char ∗Buf, char const ∗Env, unsigned32 Bufsize)

  Copies an environment value into a user-supplied buffer.
- signed32 hpss_EnvLocalUpdate (char const ∗EnvStr)

  Update an environment table entry within the local process.
- void hpss_EnvReInitialize (void)

  Reinitialize the environment variables from the override file.
- char ∗ hpss_Getenv (char const ∗Env)

  Retrieve HPSS and system environment variables.
- char ∗ hpss_GetenvDefault (char const ∗Env)

  Get the default value for an enviornment variable.

11.72.1 Detailed Description

Read and Update HPSS Environment variables.

11.72.2 Function Documentation

11.72.2.1 char∗ hpss_GetenvDefault (char const ∗ *Env*)

Get the default value for an enviornment variable.

This function is similar to hpss_Getenv(), except that it only provides the default value of an environment variable. If the default exists but its value is NULL, a blank string is returned instead. If the default does not exist at all, a NULL is returned.

Parameters

in	*Env*	Name of environment variable

Outputs:

Returns

NULL if the environment variable does not exist in the default, or "" / value represeting the variable's default value

11.73 hpss_Getenv.h File Reference

```
#include <hpss_types.h>
```

Functions

- int32_t hpss_Envcpy (char ∗Buf, char const ∗Env, uint32_t Bufsize)

  Copies an environment value into a user-supplied buffer.
- int32_t hpss_EnvLocalUpdate (char const ∗EnvStr)

  Update an environment table entry within the local process.
- void hpss_EnvReInitialize (void)

  Reinitialize the environment variables from the override file.
- char ∗ hpss_Getenv (const char ∗Env)

  Retrieve HPSS and system environment variables.
- char ∗ hpss_GetenvDefault (const char ∗Env)

  Get the default value for an enviornment variable.

11.73.1 Function Documentation

11.73.1.1 char∗ hpss_GetenvDefault (char const ∗ *Env*)

Get the default value for an enviornment variable.

This function is similar to hpss_Getenv(), except that it only provides the default value of an environment variable. If the default exists but its value is NULL, a blank string is returned instead. If the default does not exist at all, a NULL is returned.

Parameters

in	*Env*	Name of environment variable

Outputs:

Returns

NULL if the environment variable does not exist in the default, or "" / value represeting the variable's default value

11.74 hpss_hash.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <openssl/md5.h>
#include <openssl/sha.h>
#include <pthread.h>
#include <ctype.h>
#include <zlib.h>
#include <sys/socket.h>
#include "hpss_hash.h"
#include "hpss_String.h"
```

Macros

- #define ADLER32_BASE 65521

Functions

- int hpss_HashAppend (hpss_hash_t Hash,int Character)

  Append a character to the hash stream.
- int hpss_HashAppendBuf (hpss_hash_t Hash,unsigned char ∗Buffer,unsigned int Length)

  Append a byte buffer to the hash stream.
- int hpss_HashAppendStr (hpss_hash_t Hash,char ∗String)

  Append a null-terminated C string to hash stream.
- void hpss_HashBaseInit (char ∗∗envp)

  Initialize hash hardware flags.
- hpss_hash_t hpss_HashCreate (hpss_hash_type_t Type)

  Create a hash based upon the algorithm type.
- hpss_hash_t hpss_HashDecode (unsigned char ∗Buffer,unsigned int Length)

  Decode a hash context.
- int hpss_HashDelete (hpss_hash_t Hash)

  Free a hashing engine.
- int hpss_HashDigestLength (hpss_hash_type_t Type)

  Returns the number of bytes required to hold the hash digest.
- void hpss_HashDigestToHex (unsigned char ∗digest, int digest_len, unsigned char ∗digest_str)

  Convert a hash digest to hex.
- char ∗ hpss_HashDigestToString (const unsigned char ∗Buffer,unsigned int Length)

  Convert a hash digest buffer into an hex string.

- hpss_hash_t hpss_HashDuplicate (hpss_hash_t Hash)

  Duplicate an existing hash.
- int hpss_HashEncode (hpss_hash_t Hash,unsigned char ∗Buffer,unsigned int Length)

  Encode a hash context.
- int hpss_HashExtract (hpss_hash_t Hash,void ∗Buffer,unsigned int ∗Length,hpss_hash_type_t ∗Type)

  Extract a hashing engine into a buffer.
- unsigned char ∗ hpss_HashFinish (hpss_hash_t Hash,unsigned int ∗Length)

  Finish hashing and return the result.
- int hpss_HashFinishDigest (hpss_hash_t Hash,unsigned char ∗Digest,unsigned int Length)

  Finalize a hash calculation.
- char ∗ hpss_HashFinishHex (hpss_hash_t Hash)

  Return a finished hash as a hex string.
- int hpss_HashGetHWFlags ()

  Return checksum flags.
- hpss_hash_t hpss_HashLoad (void ∗Buffer,unsigned int Length,hpss_hash_type_t Type)

  Load a hash engine from a buffer.
- int hpss_HashReset (hpss_hash_t Hash)

  Reset hash engine in preparation for a new stream.
- int hpss_HashType (hpss_hash_t Hash,hpss_hash_type_t ∗Type)

  Get the hash algorithm type.
- unsigned char ∗ hpss_HexStringToHashDigest (const char ∗HexString,unsigned int ∗Length)

  Convert a hex string into a hash digest buffer.

11.74.1 Detailed Description

Functions for generating hashes (checksums) of streams of data.

This file is for generating hashes (checksums) of streams of data. The algorithms contained herein use the OpenSSL implementation of the specification, with notable exceptions for CRC32 and Adler32, which are implemented from their specification.

The algorithms for CRC32 and Adler32 can be found for their respective RFCs.

11.74.2 Macro Definition Documentation

11.74.2.1 #define ADLER32_BASE 65521

ADLER32_BASE is the largest prime number smaller than 65536

11.75 hpss_hash.h File Reference

```
#include <sys/types.h>
#include "hpss_types.h"
#include "u_signed64.h"
#include "hpss_limits.h"
#include "hpss_errno.h"
```

Macros

- #define HPSS_CKSUM_AFALG (0x00000004)
- #define HPSS_CKSUM_SSE42 (0x00000002)
- #define HPSS_CKSUM_VPMSUM (0x00000001)

Functions

- int hpss_HashAppend (hpss_hash_t Hash,int Character)

  Append a character to the hash stream.
- int hpss_HashAppendBuf (hpss_hash_t Hash,unsigned char ∗Buffer,unsigned int Length)

  Append a byte buffer to the hash stream.
- int hpss_HashAppendStr (hpss_hash_t Hash,char ∗String)

  Append a null-terminated C string to hash stream.
- void hpss_HashBaseInit (char ∗∗envp)

  Initialize hash hardware flags.
- hpss_hash_t hpss_HashCreate (hpss_hash_type_t Type)

  Create a hash based upon the algorithm type.
- hpss_hash_t hpss_HashDecode (unsigned char ∗Buffer,unsigned int Length)

  Decode a hash context.
- int hpss_HashDelete (hpss_hash_t Hash)

  Free a hashing engine.
- int hpss_HashDigestLength (hpss_hash_type_t Type)

  Returns the number of bytes required to hold the hash digest.
- void hpss_HashDigestToHex (unsigned char ∗digest, int digest_len, unsigned char ∗digest_str)

  Convert a hash digest to hex.
- char ∗ hpss_HashDigestToString (const unsigned char ∗Buffer,unsigned int Length)

  Convert a hash digest buffer into an hex string.
- hpss_hash_t hpss_HashDuplicate (hpss_hash_t Hash)

  Duplicate an existing hash.
- int hpss_HashEncode (hpss_hash_t Hash,unsigned char ∗Buffer,unsigned int Length)

  Encode a hash context.
- int hpss_HashExtract (hpss_hash_t Hash,void ∗Buffer,unsigned int ∗Length,hpss_hash_type_t ∗Type)

  Extract a hashing engine into a buffer.
- unsigned char ∗ hpss_HashFinish (hpss_hash_t Hash,unsigned int ∗Length)

  Finish hashing and return the result.
- int hpss_HashFinishDigest (hpss_hash_t Hash,unsigned char ∗Digest,unsigned int Length)

  Finalize a hash calculation.
- char ∗ hpss_HashFinishHex (hpss_hash_t Hash)

  Return a finished hash as a hex string.
- int hpss_HashGetHWFlags ()

  Return checksum flags.
- hpss_hash_t hpss_HashLoad (void ∗Buffer,unsigned int Length,hpss_hash_type_t Type)

  Load a hash engine from a buffer.
- int hpss_HashReset (hpss_hash_t Hash)

  Reset hash engine in preparation for a new stream.
- int hpss_HashType (hpss_hash_t Hash,hpss_hash_type_t ∗Type)

  Get the hash algorithm type.
- unsigned char ∗ hpss_HexStringToHashDigest (const char ∗HexString,unsigned int ∗Length)

  Convert a hex string into a hash digest buffer.

11.76 hpss_ihashtable.h File Reference

Generic intrusive hash table.

```
#include <stdbool.h>
#include <stdio.h>
#include "hpss_ilist.h"
```

Typedefs

- typedef hpss_ilist_node_t hpss_ihashtable_node_t

Enumerations

- enum hpss_ihashtable_constants { DEFAULT_HASHTABLE_SIZE = 100, MAX_LOAD_FACTOR = 150 }

  Constant values for the intrusive hash table container.

11.76.1 Detailed Description

Generic intrusive hash table.

11.76.2 Typedef Documentation

11.76.2.1 typedef **hpss_ilist_node_t hpss_ihashtable_node_t**

A hash table node is really just an intrusive list node. But, this is an implementation detail hidden by this typedef.

11.77 hpss_ilist.h File Reference

Generic intrusive doubly-linked circular list.

```
#include <stdint.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdbool.h>
#include "hpss_types.h"
#include "hpss_ilist_low_level.h"
```

Typedefs

- typedef hpss_ilist_t ∗(∗ hpss_ilist_alloc_func_t)(hpss_reqid_t RequestID, size_t NumBytes, const char ∗FileName, const char ∗FunctionName, const int LineNumber)

  Memory allocator for custom list initialization.
- typedef void(∗ hpss_ilist_destructor_t)(hpss_ilist_node_t ∗Node)

  Node destructor.
- typedef enum
  hpss_ilist_foreach_retval hpss_ilist_foreach_retval_e

Enumeration for return types of mapped functions.

- typedef struct hpss_ilist_iterator_token ∗ hpss_ilist_iterator_token_h

  Typedef for opaque token.
- typedef hpss_ilist_foreach_retval_e(∗ hpss_ilist_mapped_func_t)(hpss_ilist_node_t ∗Node)

  Mapping function type.
- typedef struct hpss_ilist_node hpss_ilist_node_t

  Just the typedef for the following hpss_ilist_node struct.
- typedef struct hpss_ilist hpss_ilist_t

  List type.

Enumerations

- enum hpss_ilist_foreach_retval

  Enumeration for return types of mapped functions.

Functions

- int32_t hpss_ilist_append (hpss_ilist_t ∗List, hpss_ilist_node_t ∗Node)

  Add a node to the end of an existing list.
- void hpss_ilist_clear (hpss_ilist_t ∗List)

  Clear an entire list by taking out all nodes and freeing memory.
- hpss_ilist_t ∗ hpss_ilist_custom_init (hpss_ilist_destructor_t Destroy, hpss_ilist_alloc_func_t AllocFunc, hpss-_reqid_t RequestID, const char ∗FileName, const char ∗FunctionName, const int LineNumber)

  Initialize a list with a custom memory allocator.
- bool hpss_ilist_empty (hpss_ilist_t ∗List)

  Check whether a list node points to anything else.
- void hpss_ilist_foreach (hpss_ilist_t ∗List, hpss_ilist_mapped_func_t F)

  Map a function across all nodes in a list.
- void hpss_ilist_foreach_range (hpss_ilist_t ∗List, hpss_ilist_mapped_func_t F, int32_t StartIndex, int32_t End-Index)

  Map a function across all nodes in a given range of a list.
- hpss_ilist_node_t ∗ hpss_ilist_get_current (const hpss_ilist_t ∗List)

  Return the current node in the list, useful for starting iteration.
- hpss_ilist_node_t ∗ hpss_ilist_get_current_r (const hpss_ilist_iterator_token_h Token)

  Return the current node in the list, useful for starting reentrant iteration.
- hpss_ilist_node_t ∗ hpss_ilist_get_node (hpss_ilist_t ∗List, int32_t Index)

  Get a node at a given index in a list.
- int32_t hpss_ilist_get_size (const hpss_ilist_t ∗const List)

  Access the size variable in the private hpss_ilist_t struct.
- void hpss_ilist_halt_iterator (hpss_ilist_t ∗List)

  Halt iteration, reset Start, Current, and End to NULL.
- hpss_ilist_t ∗ hpss_ilist_init (hpss_ilist_destructor_t Destroy)

  Initialize a list.
- int32_t hpss_ilist_init_iterator (hpss_ilist_t ∗List)

  Init the iterator to iterate over the entire list.
- hpss_ilist_iterator_token_h hpss_ilist_init_iterator_r (hpss_ilist_t ∗List)

  Define a range of indices to include during iteration. Also initialize token for reentrant iteration.
- int32_t hpss_ilist_init_iterator_range (hpss_ilist_t ∗List, int32_t Start, int32_t End)

Init the Current, Start, and End variables in the hpss_ilist struct by giving a range of indices to iterate over.

- hpss_ilist_iterator_token_h hpss_ilist_init_iterator_range_r (hpss_ilist_t ∗List, int32_t Start, int32_t End)

  Tell the iterator what range of indices to include. Also initialize token for reentrant iteration.
- int32_t hpss_ilist_insert (hpss_ilist_t ∗List, hpss_ilist_node_t ∗Node1, hpss_ilist_node_t ∗Node2)

  Insert a node into an existing list.
- int32_t hpss_ilist_join (hpss_ilist_t ∗List1, hpss_ilist_t ∗List2)

  Attach one list to the end of another list.
- int32_t hpss_ilist_prepend (hpss_ilist_t ∗List, hpss_ilist_node_t ∗Node)

  Add a node to the start of an existing list.
- int32_t hpss_ilist_reinit_iterator (hpss_ilist_t ∗List)

  Reinitialize iteration on the entire list using the halt and initialize functions above.
- int32_t hpss_ilist_reinit_iterator_range (hpss_ilist_t ∗List, int32_t Start, int32_t End)

  Reinitialize iteration on a range of indices using the halt and initialize functions above.
- void hpss_ilist_remove (hpss_ilist_t ∗List, hpss_ilist_node_t ∗Node)

  Remove a list node from a list.

11.77.1 Detailed Description

Generic intrusive doubly-linked circular list. This list is doubly linked (has a pointer to previous and next nodes) and circular (the last node->Next points back up to the top of the list). The circular nature of the list makes both append and prepend constant time operations, whereas a non-circular list would need to linearly iterate through the entire list to append a node to the list. In order to work with this module, you must have a pre-existing data structure that contains data you'd like to store in a list. Then, add another member variable to that struct of type hpss_ilist_node_t. This is the intrusive node within your data structure. You must also create a destructor function that takes an hpss_ilist_node_t pointer, destroys it (it's up to you to decide how things are destroyed, maybe by freeing the node or using memset to zero it all out) and then returns void. The destructor function pointer may be a NULL pointer if you do not wish for nodes to ever be removed from memory by the hpss_ilist API. The following steps detail how to start using an hpss_ilist_t.

1. Build up a list by calling the hpss_ilist_init function, which will return an empty list to you. 2. Then create nodes from your data structure. 3. Add your nodes to the list with the hpss_ilist_append, hpss_ilist_insert, hpss_ilist_insert_index or hpss_ilist_prepend methods. 4. Remove nodes from the list with the hpss_ilist_remove method. This will call your destructor on the node unless your destructor is NULL. 5. Use any of the foreach or iterator methods to apply the same function to multiple nodes in a list. 6. Call hpss_ilist_clear to remove all nodes from your list and free the memory allocated to your list.

Note

An intrusive linked list is a data structure in which each list node has a Next and Prev pointer that provide the linkage. "Intrusive" means the list nodes themselves do not contain a payload. Instead, each list node "intrudes on" a separate structure, meaning a list node is a member variable of that structure. Note that it is not enough to have a pointer to a list node as the member variable in the container structure. It must be the entire list node. This container structure must be defined by the calling code and should hold the payload a developer wishes to store in addition to a linked list node of type hpss_ilist_node_t.
The maximum size of an hpss_ilist_t is INT32_MAX, which is 2,147,483,647 nodes. For lists larger than that, consider a different data structure, preferably one that has a method of accessing nodes at particular indices in constant time, such as an array.
The main advantage of an intrusive implementation is that no dynamic memory allocations occur within the module, so all list operations are guaranteed to succeed.
Below is some common terminology used in this module:

- hpss_ilist_t: an intrusive linked list structure that is meant to be opaque, meaning devs don't need to manipulate hpss_ilist_t member variables to utilize this module
- node: an object of type hpss_ilist_node_t, an element of a list
- payload: the data that a developer would like to store in a list, can be of any type the dev wants (int, bool, string, etc.)

- sentinel: a node with no payload that marks the start/end of a circular list
- container: an object of a developer-defined type that contains an intrusive hpss_ilist_node_t node and some payload. A dev must create this container struct themselves. If you have an hpss_ilist_node_t node and would like to gain access to the container it lives in, use the hpss_ilist_container macro.
- destructor: a developer defined function that tells the hpss_ilist module how to remove nodes from a list. hpss_ilist_remove calls the destructor on a node whereas hpss_ilist_clear calls the destructor iteratively on every node in the list. Commonly, destructors call memset on a node or use hpss_ilist_container to find the container of the node and free that container. Alternatively, a destructor can be a NULL pointer if developers would prefer to destroy nodes manually in the calling code or if nodes should never be destroyed after being removed from a list.

The following is an example of how to use the module

```
// your responsiblity to create a struct with payload of your choice
typedef struct myStruct
{
    char *Name;                  // This is the payload
    hpss_ilist_node_t ListNode;  // THIS INTRUSIVE NODE CAN NOT BE A POINTER
} myStruct;

// you need to make your own destructor, but here’s a generic example
void my_destructor(hpss_ilist_node_t *NodeToDestroy)
{
    // given the ilist node, get its container and free that
    myStruct *container = hpss_ilist_container(
                              NodeToDestroy, // address of node to destroy
                              myStruct, // structure type
                              ListNode // field member within myStruct that represents the intrusive node
                              );
    free(container);
}

// initialize linked list
hpss_ilist_t *List = hpss_ilist_init(my_destructor);

// put 2 nodes into the list
myStruct *newNode1 = calloc(1, sizeof(*newNode1));
newNode1 -> Name = "Bailey";
// inserts node at the end of the list
int32_t err = hpss_ilist_append(List, &newNode1->ListNode);
if(err != 0)
{
    // Log the error appending a node
    printf("Could not append another node to the list\n");
}
myStruct *newNode2 = calloc(1, sizeof(*newNode2));
newNode2 -> Name = "Chris";
err = hpss_ilist_append(List, &newNode2->ListNode);
if(err != 0)
{
    // Log the error appending a node
    printf("Could not append another node to the list\n");
}

// remove one of the nodes we added to the list
// hpss_ilist_remove calls my_destructor()
hpss_ilist_remove(List, &newNode1->ListNode);
// remove any remaining node(s) in the list
hpss_ilist_clear(List);
```

Here is what an intrusive linked list with 3 myStruct nodes would look like:

Sentinel /–>+——————————+<————————————————————————-\ | | | | | /-|–Next | | | | | Prev–|—— ——————————————————————\ | | | | | | | | | +——————————+ | | | | | | | | myStruct node1 myStruct node2 myStruct node3 | | | | _________________ _________________ _________________ | | | | | Name (payload) | | Name (payload) | | Name (payload) | | | | | | _________ | | _________ | | _________ | | | | | | | | | | | | | | | |<–|-/ | | &mdash;— —|–>| Next–|—|–|–>| Next–|—|–|–>| Next–|—|—/ &mdash;———|—|–Prev |<–|–|—|–Prev |<–|–|—|–Prev | | | |_________| | | |_________| | | |_________| | | |_________________| |_________________| |_________________|

In the above diagram, the outer rectangles are myStruct containers, the inner rectangles are hpss_ilist_node_t nodes, and Name is the payload.

Note

More detailed example file at ^/src/cs/example/ilist_example.c

---

11.77.2 Typedef Documentation

11.77.2.1 typedef enum hpss_ilist_foreach_retval hpss_ilist_foreach_retval_e

Enumeration for return types of mapped functions.

When creating a mapped function, a programmer should define whether the mapping iteration should continue or not

Note

If you want your function to be mapped across an entire list, always return KEEP_ITERATING
If you want your function to only be mapped across nodes until you reach a certain constraint, use STOP_ITERATING as the return value once you have reached that constraint (For example, only map this function to nodes until we reach a node with a specific desired payload, or only map this function to the first n nodes (can also use hpss_ilist_foreach_range() for this))
0 can be used instead of STOP_ITERATING and 1 can be used in place of KEEP_ITERATING if the calling code prefers to use integers.

11.77.3 Enumeration Type Documentation

11.77.3.1 enum hpss_ilist_foreach_retval

Enumeration for return types of mapped functions.

When creating a mapped function, a programmer should define whether the mapping iteration should continue or not

Note

If you want your function to be mapped across an entire list, always return KEEP_ITERATING
If you want your function to only be mapped across nodes until you reach a certain constraint, use STOP_ITERATING as the return value once you have reached that constraint (For example, only map this function to nodes until we reach a node with a specific desired payload, or only map this function to the first n nodes (can also use hpss_ilist_foreach_range() for this))
0 can be used instead of STOP_ITERATING and 1 can be used in place of KEEP_ITERATING if the calling code prefers to use integers.

11.78 hpss_irbtree.c File Reference

Generic intrusive red-black tree.

```
#include <stdio.h>
#include <stdlib.h>
#include "hpss_irbtree.h"
```

Macros

- #define LEFT 0

  Index for left child.
- #define RIGHT 1

  Index for right child.

Typedefs

- typedef enum hpss_irbtree_color hpss_irbtree_color_t

  Node colors.

Enumerations

- enum hpss_irbtree_color { IRBTREE_RED = 0, IRBTREE_BLACK = 1 }

  Node colors.

Functions

- void hpss_irbtree_clear (hpss_irbtree_t ∗Tree, hpss_irbtree_node_destructor_t Destructor)

  Destroy an entire tree.
- int hpss_irbtree_empty (const hpss_irbtree_t ∗Tree)

  Check if a tree is empty.
- hpss_irbtree_node_t ∗ hpss_irbtree_find (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node)

  Find a node in a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_find_cmp (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node, hpss_irbtree_node_comparator_t Comparator, int KeyCompare)

  Find a node in a tree.
- int hpss_irbtree_foreach (const hpss_irbtree_t ∗Tree, hpss_irbtree_node_callback_t Callback, void ∗CallbackArg)

  Call a function on each node in a tree.
- int hpss_irbtree_foreach_node (const hpss_irbtree_node_t ∗Begin, const hpss_irbtree_node_t ∗End, hpss_-irbtree_node_callback_t Callback, void ∗CallbackArg)

  Call a function on a range of nodes by forward iteration.
- int hpss_irbtree_foreach_node_prev (const hpss_irbtree_node_t ∗RBegin, const hpss_irbtree_node_t ∗R-End, hpss_irbtree_node_callback_t Callback, void ∗CallbackArg)

  Call a function on a range of nodes by reverse iteration.
- void hpss_irbtree_init (hpss_irbtree_t ∗Tree, hpss_irbtree_node_comparator_t Comparator)

  Initialize a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_insert (hpss_irbtree_t ∗Tree, hpss_irbtree_node_t ∗Node)

  Insert a node into a tree.
- void hpss_irbtree_insert_multi (hpss_irbtree_t ∗Tree, hpss_irbtree_node_t ∗Node)

  Insert a (possibly) duplicate node into a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_lower_bound (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node)

  Find the lower bound for a key.
- hpss_irbtree_node_t ∗ hpss_irbtree_max (const hpss_irbtree_t ∗Tree)

  Get the right-most object in the tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_min (const hpss_irbtree_t ∗Tree)

  Get the left-most object in the tree.

- hpss_irbtree_node_t ∗ hpss_irbtree_node_next (const hpss_irbtree_node_t ∗Node)

    Iterate to the next node in the tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_node_prev (const hpss_irbtree_node_t ∗Node)

    Iterate to the previous node in the tree.
- void hpss_irbtree_print (const hpss_irbtree_t ∗Tree, hpss_irbtree_node_printer_t Printer)

    Call a printer function on each node in a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_remove (hpss_irbtree_t ∗Tree, hpss_irbtree_node_t ∗Node, hpss_irbtree_node_destructor_t Destructor)

    Remove a node from a tree.
- size_t hpss_irbtree_size (const hpss_irbtree_t ∗Tree)

    Get number of nodes in the tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_upper_bound (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node)

    Find the upper bound for a key.

11.78.1 Detailed Description

Generic intrusive red-black tree. The main advantage of an intrusive implementation is that no dynamic memory allocations occur, so all tree operations are guaranteed to succeed.

A red-black tree is a mostly-balanced binary tree, implemented here specifically as a binary search tree. It has the following constraints:

1. A node must be either red or black.
2. The root must be black (in our implementation).
3. All sentinel (leaf) nodes are black.
4. Every red node must have two black child nodes; alternatively, a red node must not have a red child node.
5. Every path from a given node to any descendent leaf node has the same number of black nodes along the path.

This gives the following properties:

1. The maximum depth of a node's left subtree is at most twice the maximum depth of the node's right subtree (and vice versa).
2. The maximum height of a tree is 2log(n+1), where n is the number of internal (non-leaf) nodes in the tree.
3. All tree operations (insert, remove, search) maintain O(log(n)), even is worst case.
4. All tree operations (insert, remove, search) maintain strict weak ordering.

11.79 hpss_irbtree.h File Reference

Generic intrusive red-black tree.

```
#include <stdint.h>
#include <stddef.h>
```

Classes

- struct hpss_irbtree

  A root structure for the tree. More...
- struct hpss_irbtree_node

  A node in the tree. More...

Macros

- #define hpss_irbtree_container(ptr, type, member) ((type∗)(((char∗)ptr) - offsetof(type, member)))

  Get the container of a node.

Typedefs

- typedef int(∗ hpss_irbtree_node_callback_t)(const hpss_irbtree_node_t ∗Node, void ∗Arg)

  Node callback.
- typedef int(∗ hpss_irbtree_node_comparator_t)(const hpss_irbtree_node_t ∗lhs, const hpss_irbtree_node_t ∗rhs)

  Node comparator.
- typedef void(∗ hpss_irbtree_node_destructor_t)(hpss_irbtree_node_t ∗Node)

  Node destructor.
- typedef void(∗ hpss_irbtree_node_printer_t)(const hpss_irbtree_node_t ∗Node)

  Callback to print a node.
- typedef struct hpss_irbtree_node hpss_irbtree_node_t

  Typedef for struct hpss_irbtree_node.
- typedef struct hpss_irbtree hpss_irbtree_t

  Typedef for struct hpss_irbtree.

Functions

- void hpss_irbtree_clear (hpss_irbtree_t ∗Tree, hpss_irbtree_node_destructor_t Destructor)

  Destroy an entire tree.
- int hpss_irbtree_empty (const hpss_irbtree_t ∗Tree)

  Check if a tree is empty.
- hpss_irbtree_node_t ∗ hpss_irbtree_find (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node)

  Find a node in a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_find_cmp (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node, hpss_irbtree_node_comparator_t Comparator, int KeyCompare)

  Find a node in a tree.
- int hpss_irbtree_foreach (const hpss_irbtree_t ∗Tree, hpss_irbtree_node_callback_t Callback, void ∗CallbackArg)

  Call a function on each node in a tree.
- int hpss_irbtree_foreach_node (const hpss_irbtree_node_t ∗Begin, const hpss_irbtree_node_t ∗End, hpss_-irbtree_node_callback_t Callback, void ∗CallbackArg)

  Call a function on a range of nodes by forward iteration.
- int hpss_irbtree_foreach_node_prev (const hpss_irbtree_node_t ∗RBegin, const hpss_irbtree_node_t ∗R-End, hpss_irbtree_node_callback_t Callback, void ∗CallbackArg)

  Call a function on a range of nodes by reverse iteration.
- void hpss_irbtree_init (hpss_irbtree_t ∗Tree, hpss_irbtree_node_comparator_t Comparator)

  Initialize a tree.

- hpss_irbtree_node_t ∗ hpss_irbtree_insert (hpss_irbtree_t ∗Tree, hpss_irbtree_node_t ∗Node)

  Insert a node into a tree.
- void hpss_irbtree_insert_multi (hpss_irbtree_t ∗Tree, hpss_irbtree_node_t ∗Node)

  Insert a (possibly) duplicate node into a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_lower_bound (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node)

  Find the lower bound for a key.
- hpss_irbtree_node_t ∗ hpss_irbtree_max (const hpss_irbtree_t ∗Tree)

  Get the right-most object in the tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_min (const hpss_irbtree_t ∗Tree)

  Get the left-most object in the tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_node_next (const hpss_irbtree_node_t ∗Node)

  Iterate to the next node in the tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_node_prev (const hpss_irbtree_node_t ∗Node)

  Iterate to the previous node in the tree.
- void hpss_irbtree_print (const hpss_irbtree_t ∗Tree, hpss_irbtree_node_printer_t Printer)

  Call a printer function on each node in a tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_remove (hpss_irbtree_t ∗Tree, hpss_irbtree_node_t ∗Node, hpss_-irbtree_node_destructor_t Destructor)

  Remove a node from a tree.
- size_t hpss_irbtree_size (const hpss_irbtree_t ∗Tree)

  Get number of nodes in the tree.
- hpss_irbtree_node_t ∗ hpss_irbtree_upper_bound (const hpss_irbtree_t ∗Tree, const hpss_irbtree_node_t ∗Node)

  Find the upper bound for a key.

11.79.1 Detailed Description

Generic intrusive red-black tree. The main advantage of an intrusive implementation is that no dynamic memory allocations occur, so all tree operations are guaranteed to succeed.

11.80 hpss_Mech.c File Reference

Authentication and authorization routines.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hpss_types.h"
#include "hpss_errno.h"
#include "hpss_String.h"
#include "hpss_rpc.h"
#include "hpss_Getenv.h"
#include "hpss_mech.h"
```

Functions

- signed32 hpss_GetAuthType (hpss_authn_mech_t AuthMech, hpss_rpc_auth_type_t ∗AuthType)

  Get the default authenticator type for the specified authentication mechanism.

11.80.1 Detailed Description

Authentication and authorization routines.

11.80.2 Function Documentation

11.80.2.1 signed32 hpss_GetAuthType (hpss_authn_mech_t *AuthMech*, hpss_rpc_auth_type_t ∗ *AuthType*)

Get the default authenticator type for the specified authentication mechanism.

Parameters

in	*AuthMech*	Authentication mechanism • hpss_authn_mech_krb5 (Kerberos) • hpss_authn_mech_unix (Unix)
out	*AuthType*	Authenticator type • hpss_rpc_auth_type_keytab (Keytab) Get the default authenticator type, *AuthType*, for the specified authentication mechanism, *AuthMech*.

Return values

HPSS_E_NOERROR	Success
HPSS_ENOTSUPPORTED	The authentication mechanism is not supported

11.81 hpss_net.c File Reference

HPSS Networking functions for IPV4 and IPV6.

```
#include "hpss_Getenv.h"
#include "hpss_gssapi.h"
#include "hpss_net.h"
```

Functions

- signed32 __hpss_net_maskaddr32 (const hpss_sockaddr_t ∗addr, signed32 mask)
  Retrieve a 32-bit masked socket address.
- int __hpss_net_setscopeid (hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)
  Update sin6_scope_id for link local addresses.
- int hpss_net_accept (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)
  Takes a connection request from the queue and creates a new socket.
- int hpss_net_addrmatch (const hpss_sockaddr_t ∗Address, const hpss_sockaddr_t ∗Mask, const hpss_-sockaddr_t ∗Entry)
  Compare addresses using netmask.
- int hpss_net_bind (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)
  Bind an unnamed socket to an address.
- int hpss_net_connect (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

Forms a network connection between a socket and an address.
- void hpss_net_generatemask (const hpss_sockaddr_t ∗source, hpss_sockaddr_t ∗mask, int bits)

  Generate a netmask for a given number of significant bits.
- int hpss_net_get_family_option (char ∗errbuf, size_t errbuflen)

  Retrieves the HPSS_NET_FAMILY environment variable.
- char ∗ hpss_net_get_family_string (void)

  Get the environment variable HPSS_NET_FAMILY string.
- int hpss_net_getaddrinfo (const char ∗hostname, const char ∗service, int flags, hpss_ipproto_t protocol, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Return a desired socket address based upon the HPSS_NET_FAMILY setting.
- int hpss_net_getaddrinfo_ex (const char ∗hostname, const char ∗service, int flags, hpss_ipproto_t protocol, hpss_sockaddr_t ∗∗addrs, int ∗addrcnt, char ∗name, int namelen, char ∗errbuf, size_t errbuflen)

  Return a list of socket address structures based upon HPSS_NET_FAMILY.
- void ∗ hpss_net_getinaddr (const hpss_sockaddr_t ∗addr, unsigned int ∗addrfamily, socklen_t ∗addrlen)

  Return a pointer to the internal sockaddr structure for the protocol.
- int hpss_net_getnameinfo (const hpss_sockaddr_t ∗addr, char ∗host, size_t hostlen, char ∗serv, size_t servlen, int flags, char ∗errbuf, size_t errbuflen)

  Return host and service string for provided socket address.
- int hpss_net_getpeername (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Return a socket address for a remote peer.
- int hpss_net_getport (const hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Return the port associated with a socket address.
- int hpss_net_getsockname (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Retrieve a socket address for the local side of the connection from a socket descriptor.
- int hpss_net_getuniversaladdress (const hpss_sockaddr_t ∗addr, char ∗buf, int buflen, char ∗errbuf, size_t errbuflen)

  Return the string representation of the socket address structure.
- void hpss_net_initaddr (hpss_sockaddr_t ∗addr)

  Initializes a hpss_sockaddr_t.
- int hpss_net_listen (int sockfd, int backlog, char ∗errbuf, size_t errbuflen)

  Allows a socket to be used to accept incoming connections.
- int hpss_net_setport (hpss_sockaddr_t ∗addr, unsigned short port, char ∗errbuf, size_t errbuflen)

  Update the port associated with a socket address.
- int hpss_net_socket (hpss_sockaddr_t ∗addr, int type, int protocol, char ∗errbuf, size_t errbuflen)

  Open a new socket descriptor.
- void hpss_net_strerror (int errnum, char ∗buf, size_t buflen)

  Fill in the buffer with error string.
- unsigned short hpss_net_universaladdresstoport (const char ∗address)

  Return the port from the universal address representation of the address structure.

11.81.1 Detailed Description

HPSS Networking functions for IPV4 and IPV6.

11.81.2 Function Documentation

11.81.2.1 signed32 __hpss_net_maskaddr32 (const hpss_sockaddr_t ∗ *addr,* signed32 *mask*)

Retrieve a 32-bit masked socket address.

Parameters

in	*addr*	socket address structure
in	*mask*	mask to use

Deprecated 7.4.1

Note

This function should not be used.

11.81.2.2 int __hpss_net_setscopeid (hpss_sockaddr_t ∗ *addr*, char ∗ *errbuf*, size_t *errbuflen*)

Update sin6_scope_id for link local addresses.

Parameters

in,out	*addr*	Socket address structure
out	*errbuf*	Buffer for more detailed error info
in	*errbuflen*	Length of errbuf

When using the bind() or connect() system calls on Linux, the system returns EINVAL for IPv6 link-local addresses (LLA) when the scope identifier is not defined. If the address passed in is an LLA, this function enumerates all the configured network interfaces in the system and compares the address and netmask to the hpss_sockaddr_t passed in. If there is a match, the sin6_scope_id is updated.

Note

This does not seem to be a problem on AIX since their neighbor discovery algorithm seems to be able to properly choose the interface to use for LLAs.

11.81.2.3 int hpss_net_getaddrinfo (const char ∗ *hostname*, const char ∗ *service*, int *flags*, hpss_ipproto_t *protocol*, hpss_sockaddr_t ∗ *addr*, char ∗ *errbuf*, size_t *errbuflen*)

Return a desired socket address based upon the HPSS_NET_FAMILY setting.

Parameters

in	*hostname*	Hostname to get the IP address for
in	*service*	Service to get port for
in	*flags*	Flags to send to getaddrinfo
in	*protocol*	Protocol that should be used
out	*addr*	Socket address structure
out	*errbuf*	Error buffer for more detailed error info
in	*errbuflen*	Length of errbuf

hpss_net_getaddrinfo() behaves similarly to getaddrinfo(3). It, however, does not return a list of socket address structures. Instead it tries to return a desired socket address based on the HPSS_NET_FAMILY environment variable. If ipv4 is desired, then only an IPv4 socket address will be returned. If ipv6 is desired, then it will try its best to return an IPv6 address, but if there is not one for the specified host/service, then it may return an IPv4 address. If ipv6_only is desired, then only an IPv6 address will be returned.

The protocol parameter needs to be one of: HPSS_IPPROTO_UNSPECIFIED, HPSS_IPPROTO_TCP, or HPSS_-IPPROTO_UDP. Specifying HPSS_IPPROTO_UNSPECIFIED for the protocol indicates that socket addresses with any protocol can be returned by hpss_net_getaddrinfo().

Return values

0	Success
EAI_SYSTEM	Check errno for more details, if desired
Other	EAI error code

Note

getaddrinfo() returns non-POSIX error codes in general. In order to decode the error gai_strerror() should be used.

11.81.2.4 int hpss_net_getaddrinfo_ex (const char ∗ *hostname*, const char ∗ *service*, int *flags*, hpss_ipproto_t *protocol*, hpss_sockaddr_t ∗∗ *addrs*, int ∗ *addrcnt*, char ∗ *name*, int *namelen*, char ∗ *errbuf*, size_t *errbuflen*)

Return a list of socket address structures based upon HPSS_NET_FAMILY.

Parameters

`in`	*hostname*	hostname to get the ip address for
`in`	*service*	service to get the port for
`in`	*flags*	Flags to send to getaddrinfo
`in`	*protocol*	Protocol that should be used
`out`	*addrs*	Socket address structure array
`out`	*addrcnt*	Count of addresses returned
`out`	*name*	Canonical name (if requested)
`in`	*namelen*	Storage length of name
`out`	*errbuf*	Buffer for more detailed error info
`in`	*errbuflen*	Length of errbuf

hpss_net_getaddrinfo_ex() behaves similarly to getaddrinfo(3). It, however, returns a list of socket address structures. If ipv4_only is desired, then only IPv4 addresses will be returned. If ipv6 is desired, then it will return all addresses. If ipv6_only is desired, then only IPv6 addresses will be returned.

The protocol parameter needs to be one of: HPSS_IPPROTO_UNSPECIFIED, HPSS_IPPROTO_TCP, or HPSS_-IPPROTO_UDP. Specifying HPSS_IPPROTO_UNSPECIFIED for the protocol indicates that socket addresses with any protocol can be returned by hpss_net_getaddrinfo().

Return values

0	Success
EAI_SYSTEM	Check errno for more detail if desired
Other	EAI error code

Note

getaddrinfo() returns non-POSIX error codes in general. In order to decode the error gai_strerror() should be used.
The addresses returned must be free(3)'d after use.

11.81.2.5 int hpss_net_getsockname (int *sockfd*, hpss_sockaddr_t ∗ *addr*, char ∗ *errbuf*, size_t *errbuflen*)

Retrieve a socket address for the local side of the connection from a socket descriptor.

Parameters

in	*sockfd*	Socket to get the address from
out	*addr*	Socket address structure that will contain local information
out	*errbuf*	Buffer for more detailed error information
in	*errbuflen*	Length of errbuf

hpss_net_getsockname() behaves exactly like getsockname(2), but instead with a hpss_sockaddr_t. It takes the socket descriptor, passed in and returns a socket address structure that represents the local side of a connection.

Return values

0	Success
Negative	Negated POSIX error code

11.82 hpss_net.h File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>
#include <errno.h>
#include "hpss_Getenv.h"
#include "hpssclntaddr.h"
```

Functions

- signed32 __hpss_net_maskaddr32 (const hpss_sockaddr_t ∗addr, signed32 mask)

  Retrieve a 32-bit masked socket address.
- int __hpss_net_setscopeid (hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Update sin6_scope_id for link local addresses.
- int hpss_net_accept (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Takes a connection request from the queue and creates a new socket.
- int hpss_net_addrmatch (const hpss_sockaddr_t ∗Address, const hpss_sockaddr_t ∗Mask, const hpss_-sockaddr_t ∗Entry)

  Compare addresses using netmask.
- int hpss_net_bind (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Bind an unnamed socket to an address.
- int hpss_net_connect (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Forms a network connection between a socket and an address.
- void hpss_net_generatemask (const hpss_sockaddr_t ∗source, hpss_sockaddr_t ∗mask, int bits)

  Generate a netmask for a given number of significant bits.
- int hpss_net_get_family_option (char ∗errbuf, size_t errbuflen)

  Retrieves the HPSS_NET_FAMILY environment variable.
- char ∗ hpss_net_get_family_string (void)

  Get the environment variable HPSS_NET_FAMILY string.
- int hpss_net_getaddrinfo (const char ∗hostname, const char ∗service, int flags, hpss_ipproto_t protocol, hpss-_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

Return a desired socket address based upon the HPSS_NET_FAMILY setting.

- int hpss_net_getaddrinfo_ex (const char ∗hostname, const char ∗service, int flags, hpss_ipproto_t protocol, hpss_sockaddr_t ∗∗addrs, int ∗addrcnt, char ∗name, int namelen, char ∗errbuf, size_t errbuflen)

  Return a list of socket address structures based upon HPSS_NET_FAMILY.
- void ∗ hpss_net_getinaddr (const hpss_sockaddr_t ∗addr, unsigned int ∗addrfamily, socklen_t ∗addrlen)

  Return a pointer to the internal sockaddr structure for the protocol.
- int hpss_net_getnameinfo (const hpss_sockaddr_t ∗addr, char ∗host, size_t hostlen, char ∗serv, size_t servlen, int flags, char ∗errbuf, size_t errbuflen)

  Return host and service string for provided socket address.
- int hpss_net_getpeername (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Return a socket address for a remote peer.
- int hpss_net_getport (const hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Return the port associated with a socket address.
- int hpss_net_getsockname (int sockfd, hpss_sockaddr_t ∗addr, char ∗errbuf, size_t errbuflen)

  Retrieve a socket address for the local side of the connection from a socket descriptor.
- int hpss_net_getuniversaladdress (const hpss_sockaddr_t ∗addr, char ∗buf, int buflen, char ∗errbuf, size_t errbuflen)

  Return the string representation of the socket address structure.
- void hpss_net_initaddr (hpss_sockaddr_t ∗addr)

  Initializes a hpss_sockaddr_t.
- int hpss_net_listen (int sockfd, int backlog, char ∗errbuf, size_t errbuflen)

  Allows a socket to be used to accept incoming connections.
- int hpss_net_setport (hpss_sockaddr_t ∗addr, unsigned short port, char ∗errbuf, size_t errbuflen)

  Update the port associated with a socket address.
- int hpss_net_socket (hpss_sockaddr_t ∗addr, int type, int protocol, char ∗errbuf, size_t errbuflen)

  Open a new socket descriptor.
- void hpss_net_strerror (int errnum, char ∗buf, size_t buflen)

  Fill in the buffer with error string.
- unsigned short hpss_net_universaladdresstoport (const char ∗address)

  Return the port from the universal address representation of the address structure.

11.82.1 Function Documentation

11.82.1.1 signed32 __hpss_net_maskaddr32 (const hpss_sockaddr_t ∗ *addr,* signed32 *mask*)

Retrieve a 32-bit masked socket address.

Parameters

in	*addr*	socket address structure
in	*mask*	mask to use

Deprecated 7.4.1

Note

This function should not be used.

11.82.1.2 int __hpss_net_setscopeid (hpss_sockaddr_t ∗ *addr,* char ∗ *errbuf,* size_t *errbuflen*)

Update sin6_scope_id for link local addresses.

Parameters

`in,out`	*addr*	Socket address structure
`out`	*errbuf*	Buffer for more detailed error info
`in`	*errbuflen*	Length of errbuf

When using the bind() or connect() system calls on Linux, the system returns EINVAL for IPv6 link-local addresses (LLA) when the scope identifier is not defined. If the address passed in is an LLA, this function enumerates all the configured network interfaces in the system and compares the address and netmask to the hpss_sockaddr_t passed in. If there is a match, the sin6_scope_id is updated.

Note

This does not seem to be a problem on AIX since their neighbor discovery algorithm seems to be able to properly choose the interface to use for LLAs.

11.82.1.3 int hpss_net_getaddrinfo (const char ∗ *hostname,* const char ∗ *service,* int *flags,* hpss_ipproto_t *protocol,* hpss_sockaddr_t ∗ *addr,* char ∗ *errbuf,* size_t *errbuflen*)

Return a desired socket address based upon the HPSS_NET_FAMILY setting.

Parameters

`in`	*hostname*	Hostname to get the IP address for
`in`	*service*	Service to get port for
`in`	*flags*	Flags to send to getaddrinfo
`in`	*protocol*	Protocol that should be used
`out`	*addr*	Socket address structure
`out`	*errbuf*	Error buffer for more detailed error info
`in`	*errbuflen*	Length of errbuf

hpss_net_getaddrinfo() behaves similarly to getaddrinfo(3). It, however, does not return a list of socket address structures. Instead it tries to return a desired socket address based on the HPSS_NET_FAMILY environment variable. If ipv4 is desired, then only an IPv4 socket address will be returned. If ipv6 is desired, then it will try its best to return an IPv6 address, but if there is not one for the specified host/service, then it may return an IPv4 address. If ipv6_only is desired, then only an IPv6 address will be returned.

The protocol parameter needs to be one of: HPSS_IPPROTO_UNSPECIFIED, HPSS_IPPROTO_TCP, or HPSS_-IPPROTO_UDP. Specifying HPSS_IPPROTO_UNSPECIFIED for the protocol indicates that socket addresses with any protocol can be returned by hpss_net_getaddrinfo().

Return values

0	Success
EAI_SYSTEM	Check errno for more details, if desired
Other	EAI error code

Note

getaddrinfo() returns non-POSIX error codes in general. In order to decode the error gai_strerror() should be used.

11.82.1.4 int hpss_net_getaddrinfo_ex (const char ∗ *hostname,* const char ∗ *service,* int *flags,* hpss_ipproto_t *protocol,* hpss_sockaddr_t ∗∗ *addrs,* int ∗ *addrcnt,* char ∗ *name,* int *namelen,* char ∗ *errbuf,* size_t *errbuflen*)

Return a list of socket address structures based upon HPSS_NET_FAMILY.

Parameters

in	*hostname*	hostname to get the ip address for
in	*service*	service to get the port for
in	*flags*	Flags to send to getaddrinfo
in	*protocol*	Protocol that should be used
out	*addrs*	Socket address structure array
out	*addrcnt*	Count of addresses returned
out	*name*	Canonical name (if requested)
in	*namelen*	Storage length of name
out	*errbuf*	Buffer for more detailed error info
in	*errbuflen*	Length of errbuf

hpss_net_getaddrinfo_ex() behaves similarly to getaddrinfo(3). It, however, returns a list of socket address structures. If ipv4_only is desired, then only IPv4 addresses will be returned. If ipv6 is desired, then it will return all addresses. If ipv6_only is desired, then only IPv6 addresses will be returned.

The protocol parameter needs to be one of: HPSS_IPPROTO_UNSPECIFIED, HPSS_IPPROTO_TCP, or HPSS_-IPPROTO_UDP. Specifying HPSS_IPPROTO_UNSPECIFIED for the protocol indicates that socket addresses with any protocol can be returned by hpss_net_getaddrinfo().

Return values

0	Success
EAI_SYSTEM	Check errno for more detail if desired
Other	EAI error code

Note

getaddrinfo() returns non-POSIX error codes in general. In order to decode the error gai_strerror() should be used.
The addresses returned must be free(3)'d after use.

11.82.1.5 int hpss_net_getsockname (int *sockfd,* hpss_sockaddr_t ∗ *addr,* char ∗ *errbuf,* size_t *errbuflen*)

Retrieve a socket address for the local side of the connection from a socket descriptor.

Parameters

in	*sockfd*	Socket to get the address from
out	*addr*	Socket address structure that will contain local information
out	*errbuf*	Buffer for more detailed error information
in	*errbuflen*	Length of errbuf

hpss_net_getsockname() behaves exactly like getsockname(2), but instead with a hpss_sockaddr_t. It takes the socket descriptor, passed in and returns a socket address structure that represents the local side of a connection.

Return values

0	Success
Negative	Negated POSIX error code

11.83 hpss_posix_api.h File Reference

POSIX-compliant HPSS APIs; the HPSS POSIX APIs are an unsupported set of APIs meant to act as simple POSIX wrappers around standard HPSS functions.

Functions

- int hpss_CreateWithHints (char ∗Path, mode_t Mode, hpss_cos_hints_t ∗HintsIn, hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

  Create an HPSS file with hints.
- int hpss_OpenWithHints (char ∗Path, int Oflag, mode_t Mode, hpss_cos_hints_t ∗HintsIn, hpss_cos_priorities_t ∗HintsPri, hpss_cos_hints_t ∗HintsOut)

  Open a file using HPSS hints.
- int hpss_PosixCreate (char ∗Path, mode_t Mode)

  Create an HPSS file.
- int hpss_PosixFstat (int Fildes, struct stat ∗Buf)

  Retrieve information about an open file or directory.
- long hpss_PosixLseek (int Fildes, long Offset, int Whence)

  Set the file offset for an open file handl.
- int hpss_PosixLstat (char ∗Path, struct stat ∗Buf)

  Obtain information about a file or directory, without traversing symbolic links.
- int hpss_PosixOpen (char ∗Path, int Oflag, mode_t Mode)

  Open an HPSS file.
- int hpss_PosixStat (char ∗Path, struct stat ∗Buf)

  Obtain information about a file or directory.

11.83.1 Detailed Description

POSIX-compliant HPSS APIs; the HPSS POSIX APIs are an unsupported set of APIs meant to act as simple POSIX wrappers around standard HPSS functions.

11.84 hpss_sec_api.x File Reference

HPSS security credentials.

Classes

- struct hpss_group_ids

  Security User Credentials. More...

11.84.1 Detailed Description

HPSS security credentials.

11.84.2 Class Documentation

11.84.2.1 struct hpss_group_ids

Security User Credentials.

11.85 hpss_stat.h File Reference

```
#include <sys/time.h>
#include "hpss_types.h"
```

Classes

- struct hpss_stat

  Stat structure for HPSS. More...
- struct hpss_statfs

  File system stats. More...
- struct hpss_statfs64

  File system stats (64bit) More...
- struct hpss_statvfs

  Virtual File System stat structure. More...
- struct hpss_statvfs64

  Virtual File System stat structure (64bit) More...

Typedefs

- typedef struct hpss_stat hpss_stat_t

  Stat structure for HPSS.
- typedef struct hpss_statfs64 hpss_statfs64_t

  File system stats (64bit)
- typedef struct hpss_statfs hpss_statfs_t

  File system stats.
- typedef struct hpss_statvfs64 hpss_statvfs64_t

  Virtual File System stat structure (64bit)
- typedef struct hpss_statvfs hpss_statvfs_t

  Virtual File System stat structure.

11.86 hpss_String.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdarg.h>
#include "u_signed64.h"
#include "hpss_errno.h"
#include "hpss_version.h"
#include "hpss_StringUtil.h"
#include "hpss_String.h"
#include "hpss_Getenv.h"
#include "hpss_server_types.h"
#include "hpss_server_attr.h"
#include "pvl_definitions.h"
#include "pvl_drive_attr.h"
#include "pvl_request_attr.h"
#include "hpss_i18n.h"
#include "hpss_errno_text.c"
```

Typedefs

- typedef struct suffix32_s suffix32_t

  Type for Divisors and suffixes for storage byte units.
- typedef struct suffix64_s suffix64_t

  Type for Divisors and suffixes for storage byte units.

Functions

- const char ∗ hpss_AdminStateString (signed32 adminState)

  Convert the admin state to a string.
- const char ∗ hpss_AllocStateString (signed32 allocState)

  Return a string representation or the given allocation state.
- signed32 hpss_AuthenticatorTypeFromString (const char ∗AuthenticatorTypeString, hpss_rpc_auth_type_t ∗AuthenticatorType)

  Convert the authenticator type string to type.
- const char ∗ hpss_AuthenticatorTypeString (hpss_rpc_auth_type_t AuthenticatorType)

  Convert the authenticator type to a string.
- signed32 hpss_AuthnMechTypeFromString (const char ∗AuthnMechString, hpss_authn_mech_t ∗AuthnMech)

  Convert the authentication mechanism string to type.
- const char ∗ hpss_AuthnMechTypeString (hpss_authn_mech_t AuthnMech)

  Convert the authentication mechanism type to string.
- signed32 hpss_AuthzMechTypeFromString (const char ∗AuthzMechString, hpss_authz_mech_t ∗AuthzMech)

  Convert the authorization mechanism string to type.
- const char ∗ hpss_AuthzMechTypeString (hpss_authz_mech_t AuthzMech)

  Convert the authorization mechanism type to string.
- char ∗ hpss_BuildLevelString (void)

  Construct a string containing HPSS version information.
- void hpss_Bytecount32 (unsigned32 bytecount, char ∗string)

  Convert a 32-bit byte count into a storage unit based string.
- void hpss_Bytecount64 (u_signed64 bytecount, char ∗string)

  Convert a 64-bit byte count into a storage unit based string.
- char ∗∗ hpss_CopyrightStrings (void)

  Read the copyright file and return the strings.
- const char ∗ hpss_DriveFlagString (signed32 driveFlag)

  Convert the drive flags to a string.
- const char ∗ hpss_DriveStateString (signed32 driveState)

  Convert the drive state to a string.
- const char ∗ hpss_DriveTypeString (drive_type_t driveType)

  Return a string representation or the given drive type.
- const char ∗ hpss_ErrnoName (signed32 Errno)

  Convert an error number to a name string.
- int hpss_getline (FILE ∗fp, hpss_stringbuf_t ∗buf)

  getline using an hpss conformant string
- const char ∗ hpss_HashTypeString (hpss_hash_type_t HashType)

  Convert an error number to a descriptive string.

- void hpss_HexString (char ∗string, const void ∗data, size_t len)

    Construct an SQL-style hex string from input data.
- const char ∗ hpss_LabelTypeString (signed32 labelType)

    Convert the label type to a string.
- const char ∗ hpss_MediaTypeString (media_type_t mediaType)

    Return a string representation or the given media type.
- const char ∗ hpss_MountModeString (pv_mount_mode mode)

    Return a string representation of the given PV mount mode.
- const char ∗ hpss_OpStateString (signed32 opState)

    Convert the operational state to a string.
- signed32 hpss_ProtLevelFromString (const char ∗ProtLevelString, hpss_rpc_prot_level_t ∗ProtLevel)

    Convert the protection level string to type.
- const char ∗ hpss_ProtLevelString (hpss_rpc_prot_level_t ProtLevel)

    Convert the protection level to a string.
- const char ∗ hpss_PVLActivityStateString (signed32 pvlActivityState)

    Convert the PVL activity state to a string.
- const char ∗ hpss_PVLJobStateString (signed32 pvlJobState)

    Convert the PVL job state to a string.
- const char ∗ hpss_PVRTypeString (unsigned32 PVRType)

    Convert the pvr type to a string.
- const char ∗ hpss_ServerTypeString (unsigned32 ServerType)

    Convert the server type to a string.
- int hpss_string_cmp (const hpss_string_t ∗s1, const hpss_string_t ∗s2)

    Compare two hpss strings.
- void hpss_string_fromCstr (hpss_string_t ∗s, const char ∗str)

    Create an hpss string from a C String.
- void hpss_stringbuf_clear (hpss_stringbuf_t ∗buf)

    Clear out a conformant string.
- int hpss_stringbuf_cmp (const hpss_stringbuf_t ∗buf1, const hpss_stringbuf_t ∗buf2)

    Compare two conformant strings.
- void hpss_stringbuf_delBackward (hpss_stringbuf_t ∗buf, size_t ∗cursor, size_t num)

    Remove a specified number of bytes prior to a position in the string.
- void hpss_stringbuf_delForward (hpss_stringbuf_t ∗buf, size_t ∗cursor, size_t num)

    Remove a specified number of bytes beginning at position in the string.
- void hpss_stringbuf_free (hpss_stringbuf_t ∗buf)

    Free a conformant string.
- int hpss_stringbuf_fromCstr (hpss_stringbuf_t ∗buf, const char ∗s, size_t len)

    Create a conformant string from a C string.
- int hpss_stringbuf_fromFile (hpss_stringbuf_t ∗buf, FILE ∗file)

    Create a conformant string from a file.
- int hpss_stringbuf_fromString (hpss_stringbuf_t ∗buf, const hpss_string_t ∗str)

    Create a conformant string from an hpss string type.
- int hpss_stringbuf_grow (hpss_stringbuf_t ∗buf, size_t min)

    Grow a conformant string.
- int hpss_stringbuf_insert (hpss_stringbuf_t ∗buf, size_t ∗cursor, const hpss_string_t ∗str)

    Insert a string into a position within a conformant string.
- int hpss_stringbuf_printf (hpss_stringbuf_t ∗buf, const char ∗fmt,...)

    Write bytes using a format string to a conformant string.
- void hpss_StringClean (char ∗s)

    Remove leading and trailing spaces from a string.
- signed32 hpss_StringsAreEqual (const char ∗s1, const char ∗s2)

  Compare two strings for equality.
- void hpss_time_string (const time_t ∗t, char ∗s)

  Render date/time as a string.
- int hpss_Token_Add (char ∗token, hpss_Tokens_t ∗tokens)

  Add a specified token to the array of tokens.
- int hpss_Tokenize (char ∗line, size_t len, hpss_Tokens_t ∗tokens)

  Tokenize " or '-delimited C string into HPSS tokens, ignoring spaces.
- const char ∗ hpss_UsageStateString (signed32 usageState)

  Convert the usage state to a string.
- const char ∗ s32_and_map (signed32 s32, const s32_str_t ∗table, const unsigned long numEntries)

  Find a string entry whose key & the given key returns true.
- const char ∗ s32_eq_map (signed32 s32, const s32_str_t ∗table, const unsigned long numEntries)

  Find a string entry whose key matches the given key.
- const char ∗ u32_and_map (unsigned32 u32, const u32_str_t ∗table, const unsigned long numEntries)

  Find a string entry whose key matches the given key.
- const char ∗ u32_eq_map (unsigned32 u32, const u32_str_t ∗table, const unsigned long numEntries)

  Find a string entry whose key & the given key returns true.

Variables

- const s32_str_t AdminStateStrings [ ]
- const u32_str_t AuthenticatorTypeStrings [ ]
- const u32_str_t AuthnMechStrings [ ]
- const u32_str_t AuthzMechStrings [ ]
- const s32_str_t DriveFlagStrings [ ]
- const s32_str_t DriveStateStrings [ ]
- const s32_str_t LabelTypeStrings [ ]
- const unsigned long NumAdminStateStrings
- const unsigned long NumAuthenticatorTypeStrings
- const unsigned long NumAuthnMechStrings
- const unsigned long NumAuthzMechStrings
- const unsigned long NumDriveFlagStrings
- const unsigned long NumDriveStateStrings
- const unsigned long NumErrnoStrings
- const unsigned long NumLabelTypeStrings
- const unsigned long NumOpStateStrings
- const unsigned long NumProtLevelStrings
- const unsigned long NumPVLActivityStateStrings
- const unsigned long NumPVLJobStateStrings
- const unsigned long NumPVRTypeStrings
- const unsigned long NumServerSubtypeStrings
- const unsigned long NumServerTypeStrings
- const unsigned long NumUsageStateStrings
- const s32_str_t OpStateStrings [ ]
- const u32_str_t ProtLevelStrings [ ]
- const s32_str_t PVLActivityStateStrings [ ]
- const s32_str_t PVLJobStateStrings [ ]
- const u32_str_t PVRTypeStrings [ ]
- const u32_str_t ServerSubtypeStrings [ ]
- const u32_str_t ServerTypeStrings [ ]
- const s32_str_t UsageStateStrings [ ]

11.86.1 Detailed Description

Misc. String Handling Functions

11.86.2 Function Documentation

11.86.2.1 const char∗ hpss_AdminStateString (signed32 *adminState*)

Convert the admin state to a string.

Parameters

`in`	*adminState*	Admin State

Returns

The admin state as a string, or hpss_Unknown.

11.86.2.2 const char∗ hpss_AllocStateString (signed32 *allocState*)

Return a string representation or the given allocation state.

Parameters

`in`	*allocState*	Allocation State

Returns

The allocation state as a string, or hpss_Unknown.

11.86.2.3 signed32 hpss_AuthenticatorTypeFromString (const char ∗ *AuthenticatorTypeString,* hpss_rpc_auth_type_t ∗ *AuthenticatorType*)

Convert the authenticator type string to type.

Parameters

`in`	*Authenticator-TypeString*	Authenticator Type String
`in,out`	*Authenticator-Type*	Authenticator Type

Return values

HPSS_E_NOERROR	Success
HPSS_EINVAL	Invalid input string

11.86.2.4 const char∗ hpss_AuthenticatorTypeString (hpss_rpc_auth_type_t *AuthenticatorType*)

Convert the authenticator type to a string.

Parameters

`in`	*Authenticator-Type*	Authenticator Type

Returns

The authenticator type as a string, or hpss_Unknown.

11.86.2.5 signed32 hpss_AuthnMechTypeFromString (const char ∗ *AuthnMechString,* hpss_authn_mech_t ∗ *AuthnMech*)

Convert the authentication mechanism string to type.

Parameters

`in`	*AuthnMech-String*	Authentication Mechanism String
`in,out`	*AuthnMech*	Authentication Mechanism Type

Return values

HPSS_E_NOERROR	Success
HPSS_EINVAL	Invalid input string

11.86.2.6 const char∗ hpss_AuthnMechTypeString (hpss_authn_mech_t *AuthnMech*)

Convert the authentication mechanism type to string.

Parameters

`in`	*AuthnMech*	Authentication mechanism

Returns

The authentication mechanism type as a string, or hpss_Unknown.

11.86.2.7 signed32 hpss_AuthzMechTypeFromString (const char ∗ *AuthzMechString,* hpss_authz_mech_t ∗ *AuthzMech*)

Convert the authorization mechanism string to type.

Parameters

`in`	*AuthzMech-String*	Authorization Mechanism String
`in,out`	*AuthzMech*	Authorization Mechanism Type

Return values

HPSS_E_NOERROR	Success
HPSS_EINVAL	Invalid input string

11.86.2.8 const char∗ hpss_AuthzMechTypeString (hpss_authz_mech_t *AuthzMech*)

Convert the authorization mechanism type to string.

Parameters

`in`	*AuthzMech*	Authorization mechanism

Returns

The authorization mechanism type as a string, or hpss_Unknown.

11.86.2.9 void hpss_Bytecount32 (unsigned32 *bytecount,* char ∗ *string*)

Convert a 32-bit byte count into a storage unit based string.

Convert a 32-bit byte count into a storage unit based string (e.g. 1024 -> 1kB)

Parameters

`in`	*bytecount*	Byte Count
`in,out`	*string*	String buffer

Note

The string should be long enough to hold the full length of a 64-bit integer plus the two byte suffix
Unit suffixes beyond terabyte are not yet supported.

11.86.2.10 void hpss_Bytecount64 (u_signed64 *bytecount,* char ∗ *string*)

Convert a 64-bit byte count into a storage unit based string.

Convert a 64-bit byte count into a storage unit based string (e.g. 1024 -> 1kB)

Parameters

`in`	*bytecount*	Byte Count
`in,out`	*string*	String buffer

Note

The string should be long enough to hold the full length of a 64-bit integer plus the two byte suffix
Unit suffixes beyond terabyte are not yet supported.

11.86.2.11 const char∗ hpss_DriveFlagString (signed32 *driveFlag*)

Convert the drive flags to a string.

Parameters

in	*driveFlag*	Drive Flag

Returns

The drive flag as a string, or hpss_Unknown.

11.86.2.12 const char∗ hpss_DriveStateString (signed32 *driveState*)

Convert the drive state to a string.

Parameters

in	*driveState*	Drive State

Returns

The drive state as a string, or hpss_Unknown.

11.86.2.13 const char∗ hpss_DriveTypeString (drive_type_t *driveType*)

Return a string representation or the given drive type.

Parameters

in	*driveType*	HPSS Drive Type

Returns

The drive type as a string, or "Unknown drive type" if the type cannot be determined

11.86.2.14 const char∗ hpss_ErrnoName (signed32 *Errno*)

Convert an error number to a name string.

Parameters

in	*Errno*	HPSS Error Number

Returns

The defined name of an HPSS error code, or hpss_Unknown.

11.86.2.15 int hpss_getline (FILE ∗ *fp,* hpss_stringbuf_t ∗ *buf*)

getline using an hpss conformant string

Parameters

`in`	*fp*	Read-open file
`in,out`	*buf*	String buffer to read into

Return values

0	something was read (but possibly only a newline)
-1	EOF hit before anything read
other	error

11.86.2.16 const char∗ hpss_HashTypeString (hpss_hash_type_t *HashType*)

Convert an error number to a descriptive string.

Parameters

`in`	*Errno*	HPSS Error Number

Returns

The string text associated with the error code, or hpss_Unknown.

11.86.2.17 void hpss_HexString (char ∗ *string,* const void ∗ *data,* size_t *len*)

Construct an SQL-style hex string from input data.

Construct a representation of some binary data in a hexadecimal string from suitable for use in a SQL statement.

Parameters

`in,out`	*string*	String buffer to use
`in`	*data*	Data to stringify
`in`	*len*	Length of data in bytes

Note

This function is incapable of avoiding buffer overflows; ensure that the string provided is capable of holding the hex string beore use

11.86.2.18 const char∗ hpss_LabelTypeString (signed32 *labelType*)

Convert the label type to a string.

Parameters

`in`	*labelType*	Label type

Returns

The label type as a string, or hpss_Unknown.

11.86.2.19 const char∗ hpss_MediaTypeString (media_type_t *mediaType*)

Return a string representation or the given media type.

Parameters

`in`	*mediaType*	HPSS Media Type

Returns

The media type as a string, or "Unknown media type" if the type cannot be determined

11.86.2.20 const char∗ hpss_MountModeString (pv_mount_mode *mode*)

Return a string representation of the given PV mount mode.

Parameters

`in`	*mode*	PV mount mode

Returns

The PV mount mode as a string, or "unknown mount mode" if the mode is not known.

11.86.2.21 const char∗ hpss_OpStateString (signed32 *opState*)

Convert the operational state to a string.

Parameters

`in`	*opState*	Operational State

Returns

The operational state as a string, or hpss_Unknown.

11.86.2.22 signed32 hpss_ProtLevelFromString (const char ∗ *ProtLevelString,* hpss_rpc_prot_level_t ∗ *ProtLevel*)

Convert the protection level string to type.

Parameters

`in`	*ProtLevelString*	Protection Level String
`in,out`	*ProtLevel*	Protection Level

Return values

HPSS_E_NOERROR	Success
HPSS_EINVAL	Invalid input string

11.86.2.23 const char∗ hpss_ProtLevelString (hpss_rpc_prot_level_t *ProtLevel*)

Convert the protection level to a string.

Parameters

in	*ProtLevel*	Protection level

Returns

The protection level as a string, or hpss_Unknown.

11.86.2.24 const char∗ hpss_PVLActivityStateString (signed32 *pvlActivityState*)

Convert the PVL activity state to a string.

Parameters

in	*pvlActivityState*	PVL activity state

Returns

The PVL activity state as a string, or hpss_Unknown.

11.86.2.25 const char∗ hpss_PVLJobStateString (signed32 *pvlJobState*)

Convert the PVL job state to a string.

Parameters

in	*pvlJobState*	PVL job state

Returns

The PVL job state as a string, or hpss_Unknown.

11.86.2.26 const char∗ hpss_PVRTypeString (unsigned32 *PVRType*)

Convert the pvr type to a string.

Parameters

in	*PVRType*	PVR type

Returns

The pvr type as a string, or hpss_Unknown.

11.86.2.27 const char∗ hpss_ServerTypeString (unsigned32 *ServerType*)

Convert the server type to a string.

Parameters

`in`	*ServerType*	Server type

Returns

The server type as a string, or hpss_Unknown.

11.86.2.28 int hpss_string_cmp (const hpss_string_t ∗ *s1*, const hpss_string_t ∗ *s2*)

Compare two hpss strings.

Parameters

`in`	*s1*	HPSS string
`in,out`	*s2*	HPSS string

Return values

-2	s1 character < s2 character
-1	s1 length < s2 length
0	s1 == s2
1	s1 length > s2 length
2	s1 character > s2 character

11.86.2.29 void hpss_string_fromCstr (hpss_string_t ∗ *s*, const char ∗ *str*)

Create an hpss string from a C String.

Parameters

`in`	*s*	C string
`in,out`	*str*	HPSS string

11.86.2.30 void hpss_stringbuf_clear (hpss_stringbuf_t ∗ *buf*)

Clear out a conformant string.

Write a null character to the beginning of the string, set the length to zero.

Parameters

`in`	*buf*	Conformant string pointer

11.86.2.31 int hpss_stringbuf_cmp (const hpss_stringbuf_t ∗ *buf1*, const hpss_stringbuf_t ∗ *buf2*)

Compare two conformant strings.

Parameters

in	*buf1*	Conformant string pointer
in	*buf2*	Conformant string pointer

Return values

-1	buf1 $<$ buf2
0	buf1 == buf2
1	buf1 $>$ buf2

11.86.2.32 void hpss_stringbuf_delBackward (hpss_stringbuf_t ∗ *buf,* size_t ∗ *cursor,* size_t *num*)

Remove a specified number of bytes prior to a position in the string.

Parameters

in,out	*buf*	Conformant string pointer
in	*cursor*	Position inside the buf string
in	*num*	Amount to delete

Return values

0	Success
other	Error

Note

Cursor updated following this function to point at number bytes prior
This function does not do any bounds checking.

11.86.2.33 void hpss_stringbuf_delForward (hpss_stringbuf_t ∗ *buf,* size_t ∗ *cursor,* size_t *num*)

Remove a specified number of bytes beginning at position in the string.

Parameters

in,out	*buf*	Conformant string pointer
in	*cursor*	Position inside the buf string
in	*num*	Amount to delete

Return values

0	Success
other	Error

Note

Cursor is not updated following this function
This function does not do any bounds checking.

11.86.2.34 void hpss_stringbuf_free (hpss_stringbuf_t ∗ *buf*)

Free a conformant string.

Parameters

in	*buf*	Conformant string pointer

11.86.2.35 int hpss_stringbuf_fromCstr (hpss_stringbuf_t * *buf,* const char * *s,* size_t *len*)

Create a conformant string from a C string.

Parameters

in,out	*buf*	Conformant string pointer
in	*s*	C string
in	*len*	Length of s

Return values

0	Success
other	Error

11.86.2.36 int hpss_stringbuf_fromFile (hpss_stringbuf_t * *buf,* FILE * *file*)

Create a conformant string from a file.

Parameters

in,out	*buf*	Conformant string pointer
in	*file*	Open file handle

Return values

0	Success
other	Error

Note

file must be open for reading

11.86.2.37 int hpss_stringbuf_fromString (hpss_stringbuf_t * *buf,* const hpss_string_t * *str*)

Create a conformant string from an hpss string type.

Parameters

in,out	*buf*	Conformant string pointer
in	*str*	HPSS string

Return values

0	Success
other	Error

11.86.2.38 int hpss_stringbuf_grow (hpss_stringbuf_t * *buf,* size_t *min*)

Grow a conformant string.

Parameters

`in`	*buf*	Conformant string pointer
`in`	*min*	Minimum new size

Return values

ENOMEM	Out of memory
0	Success

Note

No action occurs if the length is already greater than min

11.86.2.39 int hpss_stringbuf_insert (hpss_stringbuf_t * *buf*, size_t * *cursor*, const hpss_string_t * *str*)

Insert a string into a position within a conformant string.

Parameters

`in`	*buf*	Conformant string pointer
`in,out`	*cursor*	Index to insert str into buf; cursor is updated upon success to continue pointing at its old position in the new string, rather than pointing at the inserted string.
`in`	*str*	String to insert

Return values

0	Success
other	Error

Note

Note that no action occurs if str is empty

11.86.2.40 int hpss_stringbuf_printf (hpss_stringbuf_t * *buf*, const char * *fmt*, ...)

Write bytes using a format string to a conformant string.

Parameters

`in`	*buf*	Conformant string pointer
`in`	*fmt*	printf-style format string
`in`	*...*	additional parameters to string

Return values

0	Success
other	Error

Note

Note that no action occurs if str is empty

11.86.2.41 void hpss_StringClean (char * *s*)

Remove leading and trailing spaces from a string.

Remove leading and trailing whitespace (including newlines) from a string.

Parameters

in	*s*	String to clean

11.86.2.42 signed32 hpss_StringsAreEqual (const char ∗ *s1*, const char ∗ *s2*)

Compare two strings for equality.

Compares 2 strings to determine if the match. The compare function is not case-sensitive.

Parameters

in	*s1*	First String
in	*s2*	Second String

Return values

0	Strings are not equal
1	Strings are equal

Note

The comparison is not case-sensitive for the english alphabet set.

11.86.2.43 void hpss_time_string (const time_t ∗ *t*, char ∗ *s*)

Render date/time as a string.

Render date/time as a string in ISO format: yyyy-mm-dd HH:MM:SS

Parameters

in	*t*	Time to render (or NULL to use current time)
out	*s*	String to render into

Note

s should be at least TIME_STRING_LEN bytes in length

11.86.2.44 int hpss_Token_Add (char ∗ *token*, hpss_Tokens_t ∗ *tokens*)

Add a specified token to the array of tokens.

Parameters

in	*token*	Token to add
in	*tokens*	Array of tokens

Return values

0	Success

HPSS_ENOMEM	Out of memory

11.86.2.45 int hpss_Tokenize (char ∗ *line*, size_t *len*, hpss_Tokens_t ∗ *tokens*)

Tokenize " or '-delimited C string into HPSS tokens, ignoring spaces.

Parameters

in	*line*	String to tokenize
in	*len*	Length of *line*
in,out	*tokens*	Tokens retrieved

Return values

0	Success

Note

Tokens cannot include spaces

11.86.2.46 const char∗ hpss_UsageStateString (signed32 *usageState*)

Convert the usage state to a string.

Parameters

in	*usageState*	Usage State

Returns

The usage state as a string, or hpss_Unknown.

11.86.2.47 const char ∗ s32_and_map (signed32 *s32*, const s32_str_t ∗ *table*, const unsigned long *numEntries*)

Find a string entry whose key & the given key returns true.

Parameters

in	*s32*	Key
in	*table*	Table to search
in	*numEntries*	Number of items in the table

Returns

The string found in the table, or "hpss_Unknown"

11.86.2.48 const char ∗ s32_eq_map (signed32 *s32*, const s32_str_t ∗ *table*, const unsigned long *numEntries*)

Find a string entry whose key matches the given key.

Parameters

in	*s32*	Key
in	*table*	Table to search
in	*numEntries*	Number of items in the table

Returns

The string found in the table, or "hpss_Unknown"

11.86.2.49 const char ∗ u32_and_map (unsigned32 *u32,* const u32_str_t ∗ *table,* const unsigned long *numEntries*)

Find a string entry whose key matches the given key.

Parameters

in	*u32*	Key
in	*table*	Table to search
in	*numEntries*	Number of items in the table

Returns

The string found in the table, or "hpss_Unknown"

11.86.2.50 const char ∗ u32_eq_map (unsigned32 *u32,* const u32_str_t ∗ *table,* const unsigned long *numEntries*)

Find a string entry whose key & the given key returns true.

Parameters

in	*u32*	Key
in	*table*	Table to search
in	*numEntries*	Number of items in the table

Returns

The string found in the table, or "hpss_Unknown"

11.86.3 Variable Documentation

11.86.3.1 const s32_str_t AdminStateStrings[ ]

Initial value:

```
= {
  {ST_LOCKED, "Locked"},
  {ST_UNLOCKED, "Unlocked"},
  {ST_SHUTDOWN, "Shutdown"},
  {ST_HALT, "Halt"},
  {ST_REINITIALIZE, "Reinitialize"},
  {ST_REPAIRED, "Repaired"}
}
```

Administration State Strings

11.86.3.2 const u32_str_t AuthenticatorTypeStrings[ ]

Initial value:

```
= {
   {hpss_rpc_auth_type_none, "auth_none"},
   {hpss_rpc_auth_type_keytab, "auth_keytab"},
   {hpss_rpc_auth_type_keyfile, "auth_keyfile"},
   {hpss_rpc_auth_type_key, "auth_key"},
   {hpss_rpc_auth_type_passwd, "auth_passwd"},
}
```

Authenticator type

11.86.3.3 const u32_str_t AuthnMechStrings[ ]

Initial value:

```
= {
  {hpss_authn_mech_krb5, "krb5"},
  {hpss_authn_mech_unix, "unix"},
  {hpss_authn_mech_gsi, "gsi"},
  {hpss_authn_mech_spkm, "spkm"},
}
```

Authentication mechanisms

11.86.3.4 const u32_str_t AuthzMechStrings[ ]

Initial value:

```
= {
  {hpss_authz_mech_ldap, "ldap"},
  {hpss_authz_mech_unix, "unix"},
}
```

Authorization mechanisms

11.86.3.5 const s32_str_t DriveFlagStrings[ ]

Initial value:

```
= {
  {PVL_FLAG_DRIVE_CLEAR, "Clear"},
  {PVL_FLAG_DRIVE_MODIFIED, "Modified"},
  {PVL_FLAG_DRIVE_PVR_MVR_NOTIFY, "PVR/MVR Notify"},
  {PVL_FLAG_DRIVE_PVR_NOTIFY, "PVR Notify"},
  {PVL_FLAG_DRIVE_MVR_NOTIFY, "MVR Notify"},
  {PVL_FLAG_DRIVE_ABORT_NOTIFY, "Abort Notify"},
}
```

Drive flag strings

11.86.3.6 const s32_str_t DriveStateStrings[ ]

Initial value:

```
= {
  {PVL_DRIVE_CLEAR, "Clear"},
  {PVL_DRIVE_IN_USE, "In use"},
  {PVL_DRIVE_FREE, "Free"},
  {PVL_DRIVE_DISMOUNT_PENDING, "Dismount Pending"},
  {PVL_DRIVE_DELETED, "Deleted"},
  {PVL_DRIVE_UPDATED, "Updated"}
}
```

Drive state strings

11.86.3.7 const s32_str_t LabelTypeStrings[ ]

Initial value:

```
= {
  {PVL_LABEL_GENERIC, "N/A"},
  {PVL_LABEL_NONE, "None"},
  {PVL_LABEL_FOREIGN, "Foreign"},
  {PVL_LABEL_HPSS, "HPSS"},
  {PVL_LABEL_DATA, "Data"},
  {PVL_LABEL_NON_ANSI, "non-ANSI"},
  {PVL_LABEL_NSL, "NSL"},
  {PVL_LABEL_UNKNOWN, "Unknown"},
  {PVL_LABEL_DEFER_DEFAULT, "Defer (D)"},
  {PVL_LABEL_DEFER_SCRATCH, "Defer (S)"},
  {PVL_LABEL_DEFER_OVERWRITE, "Defer (O)"}
}
```

Volume label types

11.86.3.8 const unsigned long NumAdminStateStrings

Initial value:

```
=
(sizeof( AdminStateStrings ) / sizeof( s32_str_t ))
```

Number of administration state strings

11.86.3.9 const unsigned long NumAuthenticatorTypeStrings

Initial value:

```
=
(sizeof( AuthenticatorTypeStrings ) / sizeof( u32_str_t ))
```

Number of authenticator type strings

11.86.3.10 const unsigned long NumAuthnMechStrings

Initial value:

```
=
(sizeof( AuthnMechStrings ) / sizeof( u32_str_t ))
```

Number of authentication mechanism strings

11.86.3.11 const unsigned long NumAuthzMechStrings

Initial value:

```
=
(sizeof( AuthzMechStrings ) / sizeof( u32_str_t ))
```

Number of authorization mechanism strings

11.86.3.12 const unsigned long NumDriveFlagStrings

Initial value:

```
=
(sizeof( DriveFlagStrings ) / sizeof( s32_str_t ))
```

Number of drive flag strings

11.86.3.13 const unsigned long NumDriveStateStrings

Initial value:

```
=
(sizeof( DriveStateStrings ) / sizeof( s32_str_t ))
```

Number of drive state strings

11.86.3.14 const unsigned long NumErrnoStrings

Initial value:

```
=
(sizeof( ErrnoStrings ) / sizeof( u32_str_t ))
```

Number of error strings

11.86.3.15 const unsigned long NumLabelTypeStrings

Initial value:

```
=
(sizeof( LabelTypeStrings ) / sizeof( s32_str_t ))
```

Number of volume label type strings

11.86.3.16 const unsigned long NumOpStateStrings

Initial value:

```
=
(sizeof( OpStateStrings ) / sizeof( s32_str_t ))
```

Number of operational state strings

11.86.3.17 const unsigned long NumProtLevelStrings

Initial value:

```
=
(sizeof( ProtLevelStrings ) / sizeof( u32_str_t ))
```

Number of RPC protection level strings

11.86.3.18 const unsigned long NumPVLActivityStateStrings

Initial value:

```
=
(sizeof( PVLActivityStateStrings ) / sizeof( s32_str_t ))
```

Number of PVL activity state strings

11.86.3.19 const unsigned long NumPVLJobStateStrings

Initial value:

```
=
(sizeof( PVLJobStateStrings ) / sizeof( s32_str_t ))
```

Number of PVL job state strings

11.86.3.20 const unsigned long NumPVRTypeStrings

Initial value:

```
=
(sizeof( PVRTypeStrings ) / sizeof( u32_str_t ))
```

Number of PVR type strings

11.86.3.21 const unsigned long NumServerSubtypeStrings

Initial value:

```
=
(sizeof( ServerSubtypeStrings ) / sizeof( u32_str_t ))
```

Number of server subtype strings

11.86.3.22 const unsigned long NumServerTypeStrings

Initial value:

```
=
(sizeof( ServerTypeStrings ) / sizeof( u32_str_t ))
```

Number of server strings

11.86.3.23 const unsigned long NumUsageStateStrings

Initial value:

```
=
(sizeof( UsageStateStrings ) / sizeof( s32_str_t ))
```

Number of usage state strings

11.86.3.24 const s32_str_t OpStateStrings[ ]

Initial value:

```
= {
  {ST_ENABLED, "Enabled"},
  {ST_DISABLED, "Disabled"},
  {ST_SUSPECT, "Suspect"},
  {ST_MINOR, "Minor"},
  {ST_MAJOR, "Major"},
  {ST_BROKEN, "Broken"}
}
```

Operational States

11.86.3.25 const u32_str_t ProtLevelStrings[ ]

Initial value:

```
= {
  {hpss_rpc_protect_connect, "connect"},
  {hpss_rpc_protect_pkt, "packet"},
  {hpss_rpc_protect_pkt_integ, "packet integrity"},
  {hpss_rpc_protect_pkt_privacy, "packet privacy"}
}
```

RPC protection levels

11.86.3.26 const s32_str_t PVLActivityStateStrings[ ]

Initial value:

```
= {
  {PVL_ACT_NULL, "Null"},
  {PVL_ACT_UNCOMMITTED, "Uncommitted"},
  {PVL_ACT_CART_WAIT, "Cartridge Wait"},
  {PVL_ACT_CART_ASSIGNED, "Cartridge Assigned"},
  {PVL_ACT_DRIVE_WAIT, "Drive Wait"},
  {PVL_ACT_MOUNT_PENDING, "Mount Pending"},
  {PVL_ACT_MOUNT_FAILED, "Mount Failed"},
  {PVL_ACT_MOUNTED, "Mounted"},
  {PVL_ACT_DISMOUNT_PENDING, "Dismount Pending"},
  {PVL_ACT_INJECT, "Inject"},
  {PVL_ACT_EJECT, "Eject"},
  {PVL_ACT_READING_LABEL, "Reading Label"},
  {PVL_ACT_UNLOAD_PENDING, "Unload Pending"},
  {PVL_ACT_DISMOUNTED, "Dismounted"},
  {PVL_ACT_CHECKIN, "Tape Check-In"},
  {PVL_ACT_CHECKOUT, "Tape Check-Out"},
  {PVL_ACT_SHELF_STAT, "Tape Shelf Status"}
}
```

PVL activity state strings

11.86.3.27 const s32_str_t PVLJobStateStrings[ ]

Initial value:

```
= {
  {PVL_JOB_NULL, "Null"},
  {PVL_JOB_UNCOMMITTED, "Uncommitted"},
  {PVL_JOB_CART_WAIT, "Cartridge Wait"},
  {PVL_JOB_DRIVE_WAIT, "Drive Wait"},
  {PVL_JOB_MOUNT_WAIT, "Mount Wait"},
  {PVL_JOB_MOUNTED, "Mounted"},
  {PVL_JOB_DISMOUNT_PENDING, "Dismount Pending"},
  {PVL_JOB_ABORTING, "Aborting"},
  {PVL_JOB_INJECT, "Inject"},
  {PVL_JOB_EJECT, "Eject"},
  {PVL_JOB_IN_USE, "In Use"},
  {PVL_JOB_COMPLETED, "Completed"},
  {PVL_JOB_DEFER_DISMOUNTS, "Deferred Dismount"},
  {PVL_JOB_CHECKIN, "Tape Check-In"},
  {PVL_JOB_CHECKOUT, "Tape Check-Out"},
  {PVL_JOB_SHELF_STAT, "Tape Shelf Status"},
  {PVL_JOB_IN_USE_REDUCED, "In Use Reduced"}
}
```

PVL job state strings

11.86.3.28 const u32_str_t PVRTypeStrings[ ]

Initial value:

```
= {
  {PVR_OPERATOR_SERVER_SUBTYPE, "Oper"},
  {PVR_STK_SERVER_SUBTYPE, "STK"},
  {PVR_AML_SERVER_SUBTYPE, "AML"},
  {PVR_SCSI_SERVER_SUBTYPE, "SCSI"}
}
```

PVR type strings

11.86.3.29 const u32_str_t ServerSubtypeStrings[ ]

Initial value:

```
= {
  {0, "none"},
  {SS_DISK_SERVER_SUBTYPE, "Disk Storage Server"},
  {SS_TAPE_SERVER_SUBTYPE, "Tape Storage Server"},
  {PVR_OPERATOR_SERVER_SUBTYPE, "Operator PVR"},
  {PVR_STK_SERVER_SUBTYPE, "STK PVR"},
  {PVR_AML_SERVER_SUBTYPE, "AML PVR"},
  {PVR_SCSI_SERVER_SUBTYPE, "SCSI PVR"}
}
```

Server subtypes

11.86.3.30 const u32_str_t ServerTypeStrings[ ]

Initial value:

```
= {
  {CORE_SERVER_TYPE, "Core Server"},
  {PVL_SERVER_TYPE, "PVL"},
  {PVR_SERVER_TYPE, "PVR"},
  {MVR_SERVER_TYPE, "Mover"},
  {MM_MONITOR_SERVER_TYPE, "Metadata Monitor"},
  {FTP_DAEMON_SERVER_TYPE, "FTP Daemon"},
  {PFS_DAEMON_SERVER_TYPE, "PFS Daemon"},
  {SSM_SERVER_TYPE, "SSM System Manager"},
  {STARTUP_DAEMON_SERVER_TYPE, "Startup Daemon"},
  {MPS_SERVER_TYPE, "Migration/Purge Server"},
  {LS_SERVER_TYPE, "Location Server"},
  {GK_SERVER_TYPE, "Gatekeeper"},
  {RAIT_SERVER_TYPE, "RAIT Engine"}
}
```

Server types

11.86.3.31 const s32_str_t UsageStateStrings[ ]

Initial value:

```
= {
  {ST_ACTIVE, "Active"},
  {ST_IDLE, "Idle"},
  {ST_BUSY, "Busy"},
  {ST_UNKNOWN, "Unknown"}
}
```

Usage state strings

11.87 hpss_TrashRecord.x File Reference

Classes

- struct hpss_TrashRecord

  Trashcan Entry Attributes. More...

11.87.1 Class Documentation

11.87.1.1 struct hpss_TrashRecord

Trashcan Entry Attributes.

Class Members

hpss_-distributionkey_t	BitfileHash	Bitfile Hash
hpssoid_t	BitfileId	Bitfile Id
ns_ObjHandle_t	Handle	Handle to Object in the Trashcan
u_signed64	LengthAtDelete-Time	The length of the file when it was deleted
fstring	Name[HPSS_M-AX_FILE_NAM-E]	Original File Name

u_signed64	ParentId	Original Parent Id
hpss_-distributionkey_t	ParentNsHash	Original Parent Hash
fstring	Path[HPSS_MA-X_TRASH_PAT-H]	Path back to fileset root
unsigned32	RealmId	Realm Id of the User who deleted the object
timestamp_sec-_t	TimeCreated	Original create time
timestamp_sec-_t	TimeDeleted	Time the object was deleted
timestamp_sec-_t	TimeLastRead	Original last read time
timestamp_sec-_t	TimeModified	Original last modified time
unsigned32	UID	User who deleted the object

11.88 hpss_types.x File Reference

HPSS system type definitions.

Classes

- struct hpss_cursor_id_t
- struct hpss_rpc_endpoint

Macros

- #define HPSS_OFF_t

Typedefs

- typedef unsigned32 acct_rec_t
- typedef unsigned char byte

  Byte representation.
- typedef uint16_t hpss_db_partition_t
- typedef uint16_t hpss_distributionkey_t
- typedef uint32_t hpss_id_t
- typedef hpss_uuid_t hpss_reqid_t

  An HPSS type for request IDs.
- typedef uint32_t hpss_srvr_id_t
- typedef int16_t signed16

  16-bit signed integer
- typedef int32_t signed32

  32-bit signed integer
- typedef int8_t signed8

  8-bit signed integer
- typedef unsigned long size_t
- typedef struct timespec timespec_t
- typedef unsigned32 timestamp_sec_t

Timestamp (seconds)

- typedef struct timeval timestamp_t
- typedef uint64_t u_signed64

  64-bit unsigned integer
- typedef u_char uchar
- typedef uint16_t unsigned16

  16-bit unsigned integer
- typedef uint32_t unsigned32
- typedef uint8_t unsigned8

  8-bit unsigned integer

Enumerations

- enum hpss_authn_mech_t {
  hpss_authn_mech_invalid = 0, hpss_authn_mech_krb5 = 1, hpss_authn_mech_unix = 2, hpss_authn_mech-_gsi = 3,
  hpss_authn_mech_spkm = 4 }
- enum hpss_authz_mech_t { hpss_authz_mech_invalid = 0, hpss_authz_mech_ldap = 1, hpss_authz_mech-_unix = 2 }
- enum hpss_hash_enum
- enum hpss_migr_order_type_e { hpss_migr_order_type_create_time = 0, hpss_migr_order_type_parent_id = 1 }
- enum hpss_request_state_t {
  hpss_io_req_state_queued = 0, hpss_io_req_state_ready = 1, hpss_io_req_state_active = 2, hpss_io_req_-state_complete = 3,
  hpss_io_req_state_defunct = 4 }

  Persisted I/O Request State.
- enum hpss_request_type_t
- enum hpss_rpc_auth_type_t {
  hpss_rpc_auth_type_invalid = 0, hpss_rpc_auth_type_none = 1, hpss_rpc_auth_type_keytab = 2, hpss_rpc-_auth_type_keyfile = 3,
  hpss_rpc_auth_type_key = 4, hpss_rpc_auth_type_passwd = 5 }
- enum hpss_rpc_cred_type_t
- enum hpss_rpc_prot_level_t
- enum hpss_tape_schedule_method_t { hpss_tape_schedule_unknown = 0, hpss_tape_schedule_fifo = 1, hpss_tape_schedule_offset = 2 }
- enum lbp_verify_error_enum { lbp_verify_error_none = 0, lbp_verify_error_errno = 1, lbp_verify_error_block-_guard = 2, lbp_verify_error_block_method = 3 }

Variables

- const int hpss_hash_type_last = hpss_hash_type_crc32c

11.88.1 Detailed Description

HPSS system type definitions.

11.88.2 Class Documentation

11.88.2.1 struct hpss_cursor_id_t

Generic Cursor identifier for persistent selects

Class Members

unsigned int	GenNum	Generation Number
int	Id	Cursor Identifier

11.88.2.2 struct hpss_rpc_endpoint

HPSS RPC Communications Endpoint

11.88.3 Macro Definition Documentation

11.88.3.1 #define HPSS_OFF_t

Define a 32 bit file offset for use with HPSS

11.88.4 Typedef Documentation

11.88.4.1 typedef unsigned32 acct_rec_t

Type for basic HPSS account identifier number

11.88.4.2 typedef uint16_t hpss_db_partition_t

HPSS Database Partition number

11.88.4.3 typedef uint16_t hpss_distributionkey_t

HPSS Distribution Key

11.88.4.4 typedef uint32_t hpss_id_t

HPSS identifier

11.88.4.5 typedef uint32_t hpss_srvr_id_t

HPSS server identifier type

11.88.4.6 typedef unsigned int size_t

Generate an XDR routine for size_t

11.88.4.7 typedef struct timespec timespec_t

Timespec value

11.88.4.8 typedef struct timeval timestamp_t

Timestamp value

11.88.4.9 typedef u_char uchar

Abbreviations for unsigned char

11.88.4.10 typedef uint32_t unsigned32

32-bit unsigned integer

11.88.5 Enumeration Type Documentation

11.88.5.1 enum hpss_authn_mech_t

Supported HPSS Security Authentication Mechanisms

Enumerator

hpss_authn_mech_invalid	Invalid authentication type
hpss_authn_mech_krb5	Kerberos authentication type
hpss_authn_mech_unix	Unix authentication type
hpss_authn_mech_gsi	GSI authentication type (defunct)
hpss_authn_mech_spkm	SPKM authentiation type (defunct)

11.88.5.2 enum hpss_authz_mech_t

Supported HPSS Security Authorization Mechanisms

Enumerator

hpss_authz_mech_invalid	Invalid authorization type
hpss_authz_mech_ldap	LDAP
hpss_authz_mech_unix	Unix

11.88.5.3 enum hpss_hash_enum

Supported file hashing algorithms

Note

while ecma319 and crc32c are part of this list, they are best suited for block digests, where the sizes are relatively small. crc32c and ecma319 are used for tape block digests and not allowed as file hash algorithms.

11.88.5.4 enum hpss_migr_order_type_e

HPSS Migration Order Types

Tells the type of migration ordering

Note

default is migration record create time
These values end up in metadata, so changing existing enum member's values will incur a metadata conversion.

Enumerator

hpss_migr_order_type_create_time	migr record create time
hpss_migr_order_type_parent_id	parent ID

11.88.5.5 enum hpss_request_state_t

Persisted I/O Request State.

The request states go through this state machine, in order, in the nominal case. For request recovery, see specifics.

Enumerator

hpss_io_req_state_queued	Request is in the queue
hpss_io_req_state_ready	Request is ready to run but not running
hpss_io_req_state_active	Request is running
hpss_io_req_state_complete	Request is complete
hpss_io_req_state_defunct	Request is ready for deletion

11.88.5.6 enum hpss_request_type_t

Persisted I/O Request Type

11.88.5.7 enum hpss_rpc_auth_type_t

Supported HPSS RPC Authenticator Types

Note

These must match the GAS defintions (see hpss_gssapi.h)

Enumerator

hpss_rpc_auth_type_invalid	Invalid authentication type
hpss_rpc_auth_type_none	No authentication type
hpss_rpc_auth_type_keytab	Keytab authentication type
hpss_rpc_auth_type_keyfile	Keyfile authentication type
hpss_rpc_auth_type_key	Key authentication type
hpss_rpc_auth_type_passwd	Password authentication type

11.88.5.8 enum hpss_rpc_cred_type_t

Supported HPSS RPC Credential Types

11.88.5.9 enum hpss_rpc_prot_level_t

Supported HPSS RPC Protection Levels

11.88.5.10 enum hpss_tape_schedule_method_t

Supported HPSS Tape Scheduling Types

Enumerator

hpss_tape_schedule_unknown	Unknown method (invalid)
hpss_tape_schedule_fifo	FIFO method
hpss_tape_schedule_offset	Offset-order method

11.88.5.11 enum lbp_verify_error_enum

Logical block verification error types

Enumerator

lbp_verify_error_none	No verification failure
lbp_verify_error_errno	Check errno
lbp_verify_error_block_guard	Block guard check failure error
lbp_verify_error_block_method	Invalid block protection method error

11.88.6 Variable Documentation

11.88.6.1 const int hpss_hash_type_last = hpss_hash_type_crc32c

Last defined hash type

Note

when new values are added to hpss_hash_enum this constant should be updated to reflect the change.

11.89 hpss_vattr.h File Reference

HPSS namespace object attributes.

```
#include "hpss_types.h"
#include "hpssoid.h"
#include "hpss_cos.h"
#include "ns_interface_defs.h"
```

11.89.1 Detailed Description

HPSS namespace object attributes. This file contains the definitions and constants for the hpss_vattr structure. The hpss_vattr structure is used to keep track of the attributes of an HPSS name space object.

11.90 hpss_xml.c File Reference

Functions for creating, manipulating, and parsing XML.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include "hpss_api.h"
#include "hpss_xml.h"
```

Functions

- char ∗ hpss_ChompXMLHeader (char ∗XML, char ∗Header)

  Remove the XML Header from an XML string, and optionally return it in its own string.
- int hpss_CreateXMLWhere (hpss_userattr_list_t ∗Attrs, char ∗Query, int QueryLen)

  Generates XMLExists statements.
- int hpss_CreateXQueryDelete (char ∗XPath, char ∗Query, int QueryLen)

  Generates XQuery Delete statements.
- int hpss_CreateXQueryGet (char ∗XPath, int XMLFlags, char ∗Query, int QueryLen)

  Generates XQuery Get statements.
- int hpss_CreateXQueryUpdate (char ∗XPath, char ∗Value, char ∗Query, int QueryLen)

  Generate an insert or replace command from an XPath, value, and schema name.
- int hpss_ValidXPath (char ∗XPath)

  Validates that the argument has some of the required components of an XPath statement.
- int hpss_XMLtoAttr (char ∗XMLstr, hpss_userattr_list_t ∗Attrs, int Flags)

  Convert an XML string into some number of Key/Value pairs.
- int hpss_XPathBranch (char ∗Path, char ∗Ret)

  Obtain the XPath without the leaf component.
- int hpss_XPathLeaf (char ∗Path, char ∗Ret)

  Obtain the last component of an XPath.
- int hpss_XPathRBranch (char ∗Path, char ∗Ret)

  Obtain the XPath without the root component.
- int hpss_XPathRoot (char ∗Path, char ∗Ret)

  Obtain the first component of an XPath.
- int hpss_XPathtoXML (char ∗Attr, char ∗Val, char ∗XMLstr, int Len, int Insert, int Level)

  Convert an attribute/value pair from XPath to an XML string.

11.90.1 Detailed Description

Functions for creating, manipulating, and parsing XML.

11.91 hpss_xml.h File Reference

```
#include <sys/types.h>
#include <unistd.h>
#include "core_api_def.h"
#include "hpss_errno.h"
#include "hpss_api.h"
#include "jansson.h"
```

Functions

- char ∗ hpss_ChompXMLHeader (char ∗XML, char ∗Header)

  Remove the XML Header from an XML string, and optionally return it in its own string.
- int hpss_CreateXMLWhere (hpss_userattr_list_t ∗Attrs, char ∗Query, int QueryLen)

  Generates XMLExists statements.
- int hpss_CreateXQueryDelete (char ∗XPath, char ∗Query, int QueryLen)

  Generates XQuery Delete statements.
- int hpss_CreateXQueryGet (char ∗XPath, int XMLFlags, char ∗Query, int QueryLen)

  Generates XQuery Get statements.
- int hpss_CreateXQueryUpdate (char ∗XPath, char ∗Value, char ∗Query, int QueryLen)

  Generate an insert or replace command from an XPath, value, and schema name.
- int hpss_ValidXPath (char ∗XPath)

  Validates that the argument has some of the required components of an XPath statement.
- int hpss_XMLtoAttr (char ∗XMLstr, hpss_userattr_list_t ∗Attrs, int Flags)

  Convert an XML string into some number of Key/Value pairs.
- int hpss_XPathBranch (char ∗path, char ∗ret)

  Obtain the XPath without the leaf component.
- int hpss_XPathLeaf (char ∗path, char ∗ret)

  Obtain the last component of an XPath.
- int hpss_XPathRBranch (char ∗path, char ∗ret)

  Obtain the XPath without the root component.
- int hpss_XPathRoot (char ∗path, char ∗ret)

  Obtain the first component of an XPath.
- int hpss_XPathtoXML (char ∗Attr, char ∗Val, char ∗XMLstr, int len, int insert, int level)

  Convert an attribute/value pair from XPath to an XML string.

11.92 hpsscfg_config_api.c File Reference

Functions related to parsing structured configuration files.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include "hpsscfgx_prototypes.h"
```

Functions

- hpss_cfg_stanza_t ∗ hpss_CfgFindKey (char const ∗searchStr,hpss_cfg_stanza_t ∗cfgStart,int caseFlag)

  Search for a Key value.
- hpss_cfg_stanza_t ∗ hpss_CfgFindToken (char const ∗searchToken, hpss_cfg_stanza_t ∗cfgStart, int keyOrValue, int subLevels)

  Find a token within a key.
- void hpss_CfgFree (hpss_cfg_stanza_t ∗Head)

  Free Allocated Memory for entire hpss_config parsed structure.
- int hpss_CfgParse (char const ∗cfgPath,hpss_cfg_stanza_t ∗∗Head,int ∗parseErr,int ∗errLine)

  Parse a configuration file.

11.92.1 Detailed Description

Functions related to parsing structured configuration files.

11.93 hpsscfgx_cfg_functions.c File Reference

Support functions for working with sections of the HPSS.conf file.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <time.h>
#include <netdb.h>
#include "u_signed64.h"
#include "hpss_netopt.h"
#include "hpss_config_api.h"
#include "hpsscfgx_prototypes.h"
```

Functions

- hpss_cfg_stanza_t ∗ hpsscfgx_LookupHostName (hpss_cfg_stanza_t ∗hostStanzaList, const char ∗hostName, int ignoreDomain)

  Find a substanza for a hostname.

11.93.1 Detailed Description

Support functions for working with sections of the HPSS.conf file.

11.94 hpsscfgx_GetClientInterfaces.c File Reference

Get Client Interface List from the HPSS.conf file.

```
#include <sys/types.h>
#include <sys/param.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <net/if.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <ctype.h>
#include <errno.h>
#include <sys/ioctl.h>
#include "hpsscfgx_prototypes.h"
#include "hpss_netopt.h"
#include "hpss_errno.h"
#include "hpss_pthread.h"
```

Functions

- int hpsscfgx_ConfGetClientInterfaces (const char ∗hostName, const char ∗serverName, int ∗nwIfCount, hpss_sockaddr_t ∗∗nwIfList, char ∗∗∗nwIfNames)

  Lookup client interfaces by host.

11.94.1 Detailed Description

Get Client Interface List from the HPSS.conf file.

11.95 hpsscfgx_hpssconf.c File Reference

HPSS.conf-related functions.

```
#include <sys/types.h>
#include <sys/param.h>
#include <limits.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/tcp.h>
#include <sys/stat.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <pthread.h>
#include "hpss_netopt.h"
#include "hpss_config_api.h"
#include "hpss_conv.h"
#include "hpss_errno.h"
#include "hpss_types.h"
#include "hpss_pthread.h"
#include "hpss_Getenv.h"
```

Functions

- hpss_cfg_stanza_t ∗ hpsscfgx_ConfParse (char cfgFile[])

  Parse the configuration file.
- int hpsscfgx_ConfSetDirPaths (char ∗theDirString)

  Set the Directory Path(s) for the Configuration File.
- int hpsscfgx_ConfSetFileName (char ∗theFilename)

  Set the name of the config file.
- int hpsscfgx_NetoptFindEntry (hpss_sockaddr_t ∗NetAddr, netopt_entry_t ∗∗RetEntryPtr)

  Searches through the network option table, if present to find network options configured for the specified address.
- ssize_t hpsscfgx_NetoptGetWriteSize (int SocketDescriptor, hpss_sockaddr_t ∗IpAddr)

  Return the network (TCP/IP) write size to be used for the specified connection.
- int hpsscfgx_NetoptSetSock (int Fd, hpss_sockaddr_t ∗NetAddress)

  Adjust the Socket characteristics.

11.95.1 Detailed Description

HPSS.conf-related functions. This file contains functions used to parse the HPSS.conf "Network Options" section, and functions to find or set the network options for newly created or connected sockets.

The code in this file that parses the HPSS.conf file and builds the internal network options table is original code, developed by Michael Gleicher (Gleicher Enterprises, LLC), to avoid copyright problems with using HPSS code which accomplishes the same thing.

11.96 hpsscfgx_pattern_match.c File Reference

Pattern matching functions.

```
#include <stdio.h>
#include <string.h>
```

Macros

- #define MAX_PATTERN_BUF 1024 /∗ temporary pattern buffer ∗/

  Pattern matching executive for braces "{}".

Functions

- int hpsscfgx_PatternMatch (char ∗theString, char ∗thePattern, int ∗patternError)

  Pattern match function.

11.96.1 Detailed Description

Pattern matching functions. Note that all functions herein are intended to be general in nature and not tied to any specific item such as parsing the HPSS.conf file.

11.96.2 Macro Definition Documentation

11.96.2.1 #define MAX_PATTERN_BUF 1024 /∗ temporary pattern buffer ∗/

Pattern matching executive for braces "{}".

Parameters

in	*thePattern*	Pattern to be matched
in	*theString*	candidate string to be matched
out	*patternError*	set on exit if pattern error

Matches comma-separated sets of characters enclosed within braces. Each set of characters is treated as an independent pattern. Brace levels may be nested, as long as there is a matching "} for every "{".

For example, abc{de,f∗,[g-h],wx{i,j,k}}extra_stuff would attempt to match: abcde abcf∗ abc[g-h] qbcwx{i,j,k} which would in turn expand to abcwxi abcwxj abcwxk

If an error is encountered in the pattern string, for example, a missing "}", ∗patternError is set non-zero on exit.

Return values

TRUE	Pattern match found
FALSE	No pattern match found

Note

1. This function is called recursively.

11.97 hpsscfgx_restricted_ports.c File Reference

Functions dealing with restricted TCP/IP ports.

```
#include <stdio.h>
#include <string.h>
#include <netinet/in.h>
#include "hpss_Getenv.h"
```

Macros

- #define MIN_RESTRICTED_PORT 1024

  Get the range of restricted ports that are allowed.

11.97.1 Detailed Description

Functions dealing with restricted TCP/IP ports. This file contains functions which deal with the use of restricted TCP/IP ports. It is intended for use by applications.

11.98 hpssclntaddr.x File Reference

End user client address types used in building I/O.

Classes

- struct netaddress

  Struture for passing network addresses. More...

11.98.1 Detailed Description

End user client address types used in building I/O.

11.98.2 Class Documentation

11.98.2.1 struct netaddress

Struture for passing network addresses.

11.99 ns_interface_defs.x File Reference

Define namespace types which pass through the Core Server RPC interface.

Classes

- struct DirEntryTag

  Defines the contents of an HPSS diretory entry. More...
- struct ns_ACLConfArray

  ACL entry conformant array. More...

- struct ns_ACLEntry

    Describes a Name Service ACL entry. More...
- struct ns_DirEntryConfArray

    Directory entry conformant array. More...
- struct ns_FilesetAttrs

    Contains attribute fields for a fileset. More...
- struct ns_FilesetAttrsConfArray

    Fileset Attributes conformant array. More...
- struct ns_JunctionPathEntryConfArray

    Junction Path Attributes conformant array. More...
- struct ns_RumbleEntryConfArray

    Rumble entry conformant array. More...
- struct ns_TrashcanSettings

    Core Server Trashcan Settings. More...
- struct ns_TrashEntryConfArray

    Trashcan entry conformant array. More...
- struct RumbleEntryTag

    Defines the contents of an HPSS Rumble Entry. More...
- struct TrashEntryTag

    Defines the contents of an HPSS Trashcan Entry. More...

Typedefs

- typedef uint64_t ns_FilesetAttrBits_t

11.99.1 Detailed Description

Define namespace types which pass through the Core Server RPC interface. This idl file is used to define name space types which are passed through the Core Server's RPC interface. The ".h" file produced from this idl file will be used by the Core Server and clients of the Core Server.

11.99.2 Class Documentation

11.99.2.1 struct DirEntryTag

Defines the contents of an HPSS diretory entry.

Class Members

hpss_Attrs_t	Attrs	Attributes of the directory entry
fstring	Name[HPSS_M-AX_FILE_NAM-E]	Name of the directory entry
ns_ObjHandle_t	ObjHandle	Object Handle of the directory entry
uint64_t	ObjOffset	Offset of the entry within the directory

11.99.2.2 struct ns_ACLConfArray

ACL entry conformant array.

Has a member named ACLEntry which is a structure containing:

- ACLEntry_len - The length of the array
- ACLEntry_val - The array of ns_ACLEntry_t values

11.99.2.3 struct ns_ACLEntry

Describes a Name Service 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.

Class Members

uint32_t	EntryId	Identifier.
unsigned char	EntryType	Identifies the type of this ACL entry. These correspond to the following ACL tag types: user_obj, user_obj_delegate, user, user_delegate, foreign user, foreign_user_delegate, group_obj, group_obj_delegate, group, group_delegate, foreign_group, foreign_group_delegate, other-_obj, other_obj_delegate, foreign_other, foreign_other_delegate, any_-other, any_other_delegate, mask_obj, or unauthenticated.EntryId contents.
unsigned char	Perms	Access rights.
uint32_t	RealmId	Realm Id.

11.99.2.4 struct ns_DirEntryConfArray

Directory entry conformant array.

Has a member named DirEntry which is a structure containing:

- DirEntry_len - The length of the array
- DirEntry_val - The array of ns_DirEntry_t values

11.99.2.5 struct ns_FilesetAttrs

Contains attribute fields for a fileset.

Class Members

uint64_t	Changed-RegisterBitMap	A bit vector where each bit corresponds to a changed field in the record.
uint32_t	ClassOfService	Configured COS
uint64_t	DirectoryCount	Number of directories in the fileset

uint32_t	FamilyId	Configured file family
uint64_t	FileCount	Number of files in the fileset
ns_ObjHandle_t	FilesetHandle	Object pointing to the root of the fileset
uint64_t	FilesetId	Unique fileset id for the fileset
fstring	FilesetName[H-PSS_MAX_FS_-NAME_LENGT-H]	Human readable fileset name.
uint32_t	FilesetType	Specifies the type of the fileset the attributes are for. This is a read-only field. The following constants define the fileset types: • CORE_FS_TYPE_HPSS_ONLY This fileset is an HPSS-only fileset. • CORE_FS_TYPE_ARCHIVED This fileset is a backup copy of some other fileset. • CORE_FS_TYPE_MIRRORED This fileset is a mirrored copy of some other fileset such as an XFS fileset.
uint64_t	HardLinkCount	Number of hard links in the fileset
uint64_t	JunctionCount	Number of junctions in the fileset
uint64_t	RegisterBitMap	A bit vector for which fields trigger SSM notifications.
uint32_t	StateFlags	Flags that define the state of the fileset. Valid values include: • CORE_FS_STATE_READ The fileset allows reads. • CORE_FS_STATE_WRITE The fileset allows writes. • CORE_FS_STATE_READ_WRITE The fileset allows reads and writes. • CORE_FS_STATE_DESTROYED The fileset allows no access.
uint32_t	SubSystemId	Subsystem Id
uint64_t	SymLinkCount	Number of symbolic links in the fileset
opaque	UserData[NS_F-S_MAX_USER-_DATA]	Uninterpreted data supplied by the site. This data can be ASCII, binary, or both.

11.99.2.6 struct ns_FilesetAttrsConfArray

Fileset Attributes conformant array.

Has a member named FilesetAttrsEntry which is a structure containing:

- FilesetAttrsEntry_len - The length of the array
- FilesetAttrsEntry_val - The array of ns_FilesetAttrsEntry_t values

11.99.2.7 struct ns_JunctionPathEntryConfArray

Junction Path Attributes conformant array.

Has a member named JunctionPathEntry which is a structure containing:

- JunctionPathEntry_len - The length of the array

- JunctionPathEntry_val - The array of ns_JuntionPathEntry_t values

11.99.2.8 struct ns_RumbleEntryConfArray

Rumble entry conformant array.

Has a member named RumbleEntry which is a structure containing:

- RumbleEntry_len - The length of the array
- RumbleEntry_val - The array of ns_RumbleEntry_t values

11.99.2.9 struct ns_TrashcanSettings

Core Server Trashcan Settings.

Class Members

uint32_t	AChangeIsIn-Progress	Status change is in progress?
uint64_t	Bytes-Incinerated-DuringLastBurn	The number of bytes successfully deleted from the trash during the last burn
uint64_t	FailedDeletes-DuringLastBurn	The number of failed deletions from the trash in the last burn
uint32_t	GarbageMan-Stop	Whether garbage men are being stopped
uint32_t	NumberOf-GarbageMan-Threads	The number of garbage man threads
uint64_t	NumEligible-TrashTable-Entries	Number of objects currently eligible for deletion
uint32_t	NumTrash-EntriesPer-GarbageCan	The number of trashcan entries to be processed by each garbage man per cycle
uint32_t	SecsBetween-TrashIncinerator-Burns	The number of seconds between incinerator burns (runs)
uint32_t	SecsBetween-TrashStatRuns	The number of seconds between statistics updates
uint32_t	SecsObjects-MayRemainIn-TheTrash	The number of seconds an object in the trashcan may remain before being picked up for incineration
uint64_t	SizeOfEligibile-TrashTable-Entries	Total number of bytes for all objects in the trash eligible for deletion
uint64_t	Successful-DeletesDuring-LastBurn	The number of successful deletions from the trash in the last burn

time_t	TimeOfLast-IncineratorBurn	When the incinerator last ran
time_t	TimeStructs-WereInitialized	When internal structures were initialized last
time_t	TimeTrashTable-StatsTaken	When the last statistics snapshot was taken
uint64_t	TotalBytes-Incinerated	Total bytes successfully deleted from the trash
uint64_t	TotalEntriesIn-TrashTable	Total number of objects included in the trash
uint64_t	TotalFailed-Deletes	Total failed deletes from the trash
uint64_t	TotalSuccessful-Deletes	Total successful deletes from the trash
uint64_t	TotalTrashTable-EntrySize	Total number of bytes for all objects included in the trash
uint16_t	Trashcans-Enabled	Trashcans are on?
uint32_t	TrashDumpster-Capacity	Internal Use
uint32_t	TrashDumpster-Factor	Internal Use
uint32_t	TrashIncinerator-IsCurrently-Burning	Incinerator is burning Trash?
uint32_t	TrashIncinerator-StartupDelay	How long to delay the incinerator after start-up (in seconds)
uint32_t	TrashStructs-Initialized	Internal trashcan structures Initialized?

11.99.2.10 struct ns_TrashEntryConfArray

Trashcan entry conformant array.

Has a member named TrashEntry which is a structure containing:

- TrashEntry_len - The length of the array
- TrashEntry_val - The array of ns_TrashEntry_t values

11.99.2.11 struct RumbleEntryTag

Defines the contents of an HPSS Rumble Entry.

Class Members

hpss_Rumble-Entry_t	RumbleItem	Rumble Entry Data

11.99.2.12 struct TrashEntryTag

Defines the contents of an HPSS Trashcan Entry.

Class Members

fstring	CurrentName[HPSS_MAX_FILE_NAME]	File name in the trashcan
ns_ObjHandle	TrashParentHandle	Handle to the Trashcan Directory
hpss_TrashRecord_t	TrashRecord	Trashcan Entry Data

11.99.3 Typedef Documentation

11.99.3.1 typedef uint64_t **ns_FilesetAttrBits_t**

Define the structure to hold the FilesetAttrBits.

11.100 ns_ObjHandle.x File Reference

Define the namespae object handle.

Classes

- struct ns_ObjHandle

  Name Service Object Handle. More...

11.100.1 Detailed Description

Define the namespae object handle.

11.100.2 Class Documentation

11.100.2.1 struct ns_ObjHandle

Name Service Object Handle.

Name Server Object Handles should be treated as an opaque object, and the Client API handle functions should be used in order to get specific information out in order to safeguard application compatability for future releases.

Class Members

hpss_srvr_id_t	CoreServerId	Id of Core Server that created the Handle.
uint64_t	FileId	HardLink's pointer back to original file file.
hpss_-distributionkey_t	FileNsHash	HardLink's hash for the original file
byte	Flags	Specifies a bit vector which conveys additional information. Defined flags are: • NS_OH_FLAG_FILESET_ROOT handle is for the root node of a fileset.
uint64_t	Generation	Object-id generation number.
uint64_t	ObjId	Unique object identifier.
hpss_-distributionkey_t	ObjNsHash	Hash of parent id/name that directs to particular partition
byte	Type	Object Type identifier.

11.101 ss_pvlist.x File Reference

Storage server primary data type definitions.

Classes

- struct pv_list

  List of physical volumes. More...
- struct pv_list_element

  Physical volume location information. More...

11.101.1 Detailed Description

Storage server primary data type definitions.

11.101.2 Class Documentation

11.101.2.1 struct pv_list

List of physical volumes.

Has a member named List which is a structure containing:

- List_len - The length of the array
- List_val - The array of pv_list_element_t values

11.101.2.2 struct pv_list_element

Physical volume location information.

Class Members

uint32_t	Flags	Specifies the location of the volume. The following constants may be set in this field: • PV_MOUNT_SUCCESS • PV_UNMOUNT_NOENT • PV_UNMOUNT_SUCCESS • PV_ON_SHELF
media_type_t	MediaType	Media Type
fstring	Name[HPSS_P-V_NAME_SIZE]	Physical volume name

11.102 u_signed64.h File Reference

```
#include "hpss_types.h"
```

Macros

- #define add64_3m(a, b, c)
- #define add64m(a1, a2)
- #define and64m(a1, a2)
- #define andbit64m(a, b)
- #define bld64m(a, b)
- #define cast32m(a)
- #define cast64m(a)
- #define chkbit64m(a, b)
- #define clrbit64m(a, b)
- #define CONVERT_LONGLONG_TO_U64(LL, U64)
- #define CONVERT_U64_TO_LONGLONG(U64, LL)
- #define CS_DEC64_LEN 20
- #define dec64m(a1, a2)
- #define div2x64m(a1, a2)
- #define div64_3m(a, b, c)
- #define div64m(a1, a2)
- #define eq64m(a1, a2)
- #define eqz64m(a)
- #define ge64m(a1, a2)
- #define gt64m(a1, a2)
- #define high32m(a)
- #define inc64m(a1, a2)
- #define le64m(a1, a2)
- #define low32m(a)
- #define lt64m(a1, a2)
- #define mem64m(a)
- #define mod2x64m(a1, a2)
- #define mod64_3m(a, b, c)
- #define mod64m(a1, a2)
- #define mul64_3m(a, b, c)

- #define mul64m(a1, a2)
- #define neq64m(a1, a2)
- #define neqz64m(a)
- #define not64m(a)
- #define or64m(a1, a2)
- #define orbit64m(a, b)
- #define shl64_3m(a, b, s)
- #define shl64_ipm(a, s)
- #define shl64m(a1, a2)
- #define shr64_3m(a, b, s)
- #define shr64_ipm(a, s)
- #define shr64m(a1, a2)
- #define sub64_3m(a, b, c)
- #define sub64m(a1, a2)

11.102.1 Macro Definition Documentation

11.102.1.1 #define CS_DEC64_LEN 20

String length to print 2^64 as an integer

Index

__hpss_net_maskaddr32
    hpss_net.c, 510
    hpss_net.h, 514
__hpss_net_setscopeid
    hpss_net.c, 511
    hpss_net.h, 514
64-bit Math Library, 263
    add64_3m, 264
    add64m, 264
    and64m, 264
    andbit64m, 264
    bld64m, 264
    cast32m, 264
    cast64m, 264
    chkbit64m, 264
    clrbit64m, 264
    dec64m, 265
    div2x64m, 265
    div64_3m, 265
    div64m, 265
    eq64m, 265
    eqz64m, 265
    ge64m, 265
    gt64m, 265
    high32m, 266
    inc64m, 266
    le64m, 266
    low32m, 266
    lt64m, 266
    mem64m, 266
    mod2x64m, 266
    mod64_3m, 266
    mod64m, 266
    mul64_3m, 266
    mul64m, 267
    neq64m, 267
    neqz64m, 267
    not64m, 267
    or64m, 267
    orbit64m, 267
    shl64_3m, 267
    shl64_ipm, 267
    shl64m, 267
    shr64_3m, 267
    shr64_ipm, 268
    shr64m, 268
    sub64_3m, 268
    sub64m, 268

ACLs, 170
    hpss_DeleteACL, 170
    hpss_DeleteACLHandle, 171
    hpss_GetACL, 172
    hpss_GetACLHandle, 172
    hpss_SetACL, 173
    hpss_SetACLHandle, 174
    hpss_UpdateACL, 174
    hpss_UpdateACLHandle, 175
ADLER32_BASE
    hpss_hash.c, 498
API_AddAllRegisterValues
    Client API, 85
API_AddFSDirCache
    api_cache.c, 364
API_AddHomeDirCache
    api_cache.c, 364
API_AddRegisterValues
    Client API, 85
API_AllocateRequestIOR
    api_misc.c, 396
API_ApplyRootDir
    api_chroot.c, 371
API_BuildFileSrcSink
    api_misc.c, 397
API_BuildXferSrcSink
    api_misc.c, 397
API_CONFIG_DEFAULT
    Client API, 79
API_CalcBlock32
    Data Structure Conversion, 229
API_CalcBlock64
    Data Structure Conversion, 229
API_CalcBufOffsetAndSize
    api_misc.c, 397
API_CancelDataTask
    api_misc.c, 398
API_CheckFileDescBounds
    api_misc.c, 398
API_ChecksumStateToFlags
    api_fgetattr.c, 376
API_CleanupInterruptBuffer
    api_misc.c, 399
API_ClearFSDirCache
    api_cache.c, 365
API_ClearFileTableBusy
    api_misc.c, 399
API_ClearHomeDirCache
    api_cache.c, 365
API_ClientAPIInit

api_init.c, 385
API_CloseDataConnList
api_misc.c, 399
API_CloseListenDesc
api_misc.c, 399
API_CloseThrdCntlDesc
api_misc.c, 400
API_ConvertAttrToVAttrSelFlags
api_misc.c, 400
API_ConvertAttrsToVAttrs
api_misc.c, 400
API_ConvertModeToPosixMode
Data Structure Conversion, 230
API_ConvertPosixModeToMode
Data Structure Conversion, 230
API_ConvertPosixTimeToTime
Data Structure Conversion, 230
API_ConvertTimeToPosixTime
Data Structure Conversion, 231
API_ConvertVAttrToAttrSelFlags
api_misc.c, 401
API_ConvertVAttrsToAttrs
api_misc.c, 401
API_CopyBatchByStatus
api_stage.c, 439
API_DEBUG_ERROR
Client API, 79
API_DEBUG_NONE
Client API, 79
API_DEBUG_REQUEST
Client API, 79
API_DEBUG_TRACE
Client API, 79
API_DebugPrintError
api_init.c, 385
API_DebugPrintRequest
api_init.c, 385
API_DebugPrintTrace
api_init.c, 387
API_DescListAlloc
api_misc.c, 401
API_DescListClear
api_misc.c, 401
API_DescListFree
api_misc.c, 402
API_DescListIsSet
api_misc.c, 402
API_DescListSet
api_misc.c, 402
API_DescListZero
api_misc.c, 402
API_DetermineAcct
api_acct.c, 360
API_DivideFilePath
api_misc.c, 402
API_ENABLE_LOGGING
Client API, 79
API_ExpandFileTable
api_misc.c, 403
API_FileTableSize
api_misc.c, 403
API_FillFreeList
api_misc.c, 403
API_FreeCwdStack
api_path.c, 422
API_FreeIODPtrs
api_misc.c, 403
API_FreeIOR
api_misc.c, 405
hpss_api.h, 482
API_FreeRequestIOR
api_misc.c, 405
API_GetAcctForParent
api_acct.c, 360
API_GetAllSubsystems
api_misc.c, 405
API_GetAppliedPath
api_misc.c, 405
API_GetBatchStatus
api_stage.c, 439
hpss_api.h, 483
API_GetClientAddr
api_misc.c, 407
hpss_api.h, 483
API_GetDataThreadStackSize
Client API Control, 192
API_GetFSDirCache
api_cache.c, 365
API_GetHomeDirCache
api_cache.c, 366
API_GetJunctionCache
api_cache.c, 366
API_GetMultiFileDigest
api_fgetattr.c, 377
API_GetMultiFileDigestHandle
api_fgetattr.c, 377
API_GetNextIORequestID
api_misc.c, 407
API_GetObjectFilesetId
api_misc.c, 407
API_GetOpenTableSlot
api_misc.c, 408
API_GetServerIDForSubsystem
Client API, 85
API_GetServerIDFromObjHandle
api_init.c, 387
API_GetSingleFileDigestHandle
api_fgetattr.c, 378
API_GetUniqueRequestID
Client API, 86
API_GrabConn
api_init.c, 387
API_HexDecode
api_misc.c, 408
API_HexEncode
api_misc.c, 408

API_InitCwdStack
 api_path.c, 422
API_InitFSDirCache
 api_cache.c, 366
API_InitHomeDirCache
 api_cache.c, 366
API_InitRootHandle
 api_init.c, 388
API_InitializeServerIOD
 api_misc.c, 410
API_InsertInFreeList
 api_misc.c, 410
API_InterruptDataTask
 api_misc.c, 410
API_LockCondition
 api_misc.c, 411
API_LockMutex
 api_misc.c, 411
API_MSGTYPE_MVRPROT
 Client API, 80
API_MSGTYPE_PDATA
 Client API, 80
API_ObjHandleToString
 api_misc.c, 412
API_OpenListenDesc
 api_misc.c, 412
API_OpenThrdCntlDesc
 api_misc.c, 412
API_PathFromCwdStack
 api_path.c, 423
API_PerformRetry
 api_misc.c, 414
API_PutAcctForParent
 api_acct.c, 361
API_RebuildJunctionCache
 api_cache.c, 368
API_ReleaseConn
 api_init.c, 388
API_RemoveCachedTrash
 api_misc.c, 415
API_RemoveFSDirCache
 api_cache.c, 368
API_RemoveHomeDirCache
 api_cache.c, 368
API_RemoveRegisterValues
 Client API, 86
API_RetrieveTrashcanHandle
 api_misc.c, 415
API_SetAcctID
 api_init.c, 389
API_SetCurrentGID
 api_init.c, 389
API_SetCurrentSessionAcct
 api_acct.c, 361
API_SetCwd
 api_init.c, 389
API_SetCwdStack
 api_path.c, 423
API_SetFileTableBusy
 api_misc.c, 416
API_SetLastHPSSErrno
 api_misc.c, 417
API_SetThreadCwd
 api_init.c, 389
API_SetUmask
 api_init.c, 390
API_SocketSetOptions
 Client API Control, 192
API_StageBatchFree
 api_stage.c, 440
 hpss_api.h, 483
API_StageBatchGetEntry
 Client API, 86
API_StageBatchInit
 api_stage.c, 440
 hpss_api.h, 485
API_StageBatchInsertBFHandle
 Client API, 86
API_StageBatchInsertBFObj
 Client API, 87
API_StageBatchInsertFd
 Client API, 88
API_StageStatusFree
 api_stage.c, 440
 hpss_api.h, 485
API_TRANSFER_TCP
 Client API, 80
API_TransferType
 api_init.c, 390
API_TranslateError
 api_misc.c, 417
API_TraversePath
 api_path.c, 423
API_UDAFindKey
 api_fgetattr.c, 379
API_USE_CONFIG
 Client API, 80
API_USE_SAN3P
 Client API, 80
API_UnlockCondition
 api_misc.c, 417
API_UnlockFileTableEntries
 api_misc.c, 417
API_UnlockMutex
 api_misc.c, 418
API_UpdateBatchStatus
 api_stage.c, 440
API_core_OpenBitfile
 api_wrappers.c, 450
API_net_bind
 api_misc.c, 411
Account
 hpss_Attrs, 352
Account Validation Site Library, 342
 av_site_AcctIdxToName, 342
 av_site_AcctNameToIdx, 343

av_site_Initialize, 346
av_site_Shutdown, 346
av_site_ValidateAccount, 346
av_site_ValidateChacct, 347
av_site_ValidateChown, 348
av_site_ValidateCreate, 349
acct_Site.c, 357
acct_rec_t
hpss_types.x, 547
acct_site.h, 358
Activity
hpss_Attrs, 352
add64_3m
64-bit Math Library, 264
add64m
64-bit Math Library, 264
AdminStateStrings
hpss_String.c, 537
and64m
64-bit Math Library, 264
andbit64m
64-bit Math Library, 264
api_config, 71
api_dist_file_info_t, 73
api_dist_file_info_t::file_info, 480
api_namespec, 73
api_access.c, 358
api_acct.c, 359
API_DetermineAcct, 360
API_GetAcctForParent, 360
API_PutAcctForParent, 361
API_SetCurrentSessionAcct, 361
api_acl.c, 362
api_cache.c, 363
API_AddFSDirCache, 364
API_AddHomeDirCache, 364
API_ClearFSDirCache, 365
API_ClearHomeDirCache, 365
API_GetFSDirCache, 365
API_GetHomeDirCache, 366
API_GetJunctionCache, 366
API_InitFSDirCache, 366
API_InitHomeDirCache, 366
API_RebuildJunctionCache, 368
API_RemoveFSDirCache, 368
API_RemoveHomeDirCache, 368
fsrootkeycompare, 368
stale_thresh_cmp_factor, 369
thresh_cmp, 369
userkeycompare, 369
api_chdir.c, 370
api_chmod.c, 370
api_chown.c, 371
api_chroot.c, 371
API_ApplyRootDir, 371
api_close.c, 373
api_closedir.c, 373
api_copyfile.c, 374
api_cos.c, 374
api_distfile.c, 374
api_fgetattr.c, 375
API_ChecksumStateToFlags, 376
API_GetMultiFileDigest, 377
API_GetMultiFileDigestHandle, 377
API_GetSingleFileDigestHandle, 378
API_UDAFindKey, 379
api_fileset.c, 379
api_fsetattr.c, 380
api_getcwd.c, 381
api_gettrash.c, 381
api_ids.c, 382
hpss_GetUID, 382
api_init.c, 383
API_ClientAPIInit, 385
API_DebugPrintError, 385
API_DebugPrintRequest, 385
API_DebugPrintTrace, 387
API_GetServerIDFromObjHandle, 387
API_GrabConn, 387
API_InitRootHandle, 388
API_ReleaseConn, 388
API_SetAcctID, 389
API_SetCurrentGID, 389
API_SetCwd, 389
API_SetThreadCwd, 389
API_SetUmask, 390
API_TransferType, 390
api_junction.c, 390
api_link.c, 391
api_lookup.c, 391
api_lseek.c, 392
api_misc.c, 392
API_AllocateRequestIOR, 396
API_BuildFileSrcSink, 397
API_BuildXferSrcSink, 397
API_CalcBufOffsetAndSize, 397
API_CancelDataTask, 398
API_CheckFileDescBounds, 398
API_CleanupInterruptBuffer, 399
API_ClearFileTableBusy, 399
API_CloseDataConnList, 399
API_CloseListenDesc, 399
API_CloseThrdCntlDesc, 400
API_ConvertAttrToVAttrSelFlags, 400
API_ConvertAttrsToVAttrs, 400
API_ConvertVAttrToAttrSelFlags, 401
API_ConvertVAttrsToAttrs, 401
API_DescListAlloc, 401
API_DescListClear, 401
API_DescListFree, 402
API_DescListIsSet, 402
API_DescListSet, 402
API_DescListZero, 402
API_DivideFilePath, 402
API_ExpandFileTable, 403
API_FileTableSize, 403

API_FillFreeList, 403
API_FreeIODPtrs, 403
API_FreeIOR, 405
API_FreeRequestIOR, 405
API_GetAllSubsystems, 405
API_GetAppliedPath, 405
API_GetClientAddr, 407
API_GetNextIORequestID, 407
API_GetObjectFilesetId, 407
API_GetOpenTableSlot, 408
API_HexDecode, 408
API_HexEncode, 408
API_InitializeServerIOD, 410
API_InsertInFreeList, 410
API_InterruptDataTask, 410
API_LockCondition, 411
API_LockMutex, 411
API_ObjHandleToString, 412
API_OpenListenDesc, 412
API_OpenThrdCntlDesc, 412
API_PerformRetry, 414
API_RemoveCachedTrash, 415
API_RetrieveTrashcanHandle, 415
API_SetFileTableBusy, 416
API_SetLastHPSSErrno, 417
API_TranslateError, 417
API_UnlockCondition, 417
API_UnlockFileTableEntries, 417
API_UnlockMutex, 418
API_net_bind, 411
hpss_GetIntersectedList, 418
hpss_RemoveIntersectionFromList, 418
api_misc_admin.c, 419
api_mkdir.c, 419
api_namespec_t
Client API, 82
api_open.c, 420
api_opendir.c, 421
api_path.c, 421
API_FreeCwdStack, 422
API_InitCwdStack, 422
API_PathFromCwdStack, 423
API_SetCwdStack, 423
API_TraversePath, 423
NormalizePath, 426
api_pio.c, 426
PIOInit, 427
api_posix_lseek.c, 427
api_posix_open.c, 428
api_purge.c, 428
api_purgelock.c, 429
api_purgeonmigrate.c, 429
api_rddir.c, 429
hpss_Rumble, 431
api_rdlink.c, 431
hpss_Readlink, 432
hpss_ReadlinkHandle, 432
api_rdlist.c, 433
hpss_ReadList, 433
api_read.c, 434
api_register.c, 435
api_rename.c, 435
api_rewdir.c, 436
api_rmdir.c, 436
Common_hpss_Rmdir, 437
Common_hpss_RmdirHandle, 437
api_stage.c, 438
API_CopyBatchByStatus, 439
API_GetBatchStatus, 439
API_StageBatchFree, 440
API_StageBatchInit, 440
API_StageStatusFree, 440
API_UpdateBatchStatus, 440
api_stat.c, 442
api_statfs.c, 442
api_statsubsys.c, 443
api_stream.c, 443
api_symlink.c, 444
api_trunc.c, 445
api_umask.c, 445
api_undelete.c, 446
api_unlink.c, 446
Common_hpss_Unlink, 447
api_userattr.c, 447
api_utime.c, 449
api_wrappers.c, 449
API_core_OpenBitfile, 450
api_write.c, 450
api_wrtlist.c, 451
hpss_WriteList, 452
Authentication, 198
hpss_PurgeLoginCred, 198
hpss_SetAppLoginCred, 198
hpss_SetAuditInfo, 199
hpss_SetLoginCred, 199
AuthenticatorTypeStrings
hpss_String.c, 537
AuthnMechStrings
hpss_String.c, 538
AuthzMechStrings
hpss_String.c, 538
av_site_AcctIdxToName
Account Validation Site Library, 342
av_site_AcctNameToIdx
Account Validation Site Library, 343
av_site_Initialize
Account Validation Site Library, 346
av_site_Shutdown
Account Validation Site Library, 346
av_site_ValidateAccount
Account Validation Site Library, 346
av_site_ValidateChacct
Account Validation Site Library, 347
av_site_ValidateChown
Account Validation Site Library, 348
av_site_ValidateCreate

Account Validation Site Library, 349

bf_activity_count, 453
bf_sc_attrib, 453
bf_vv_attrib, 454
bf_xattrib, 455
bfs_bitfile_obj_handle, 455
bfs_callback_addr, 455
bfs_callback_ret_msg, 456
bfs_file_hash_digest, 456
bfs_userif_def.x, 452
BitfileObj
  hpss_Attrs, 352
bld64m
  64-bit Math Library, 264
Buffered Data, 208
  hpss_Fclose, 208
  hpss_Fcntl, 209
  hpss_Fflush, 210
  hpss_Fgetc, 210
  hpss_Fgets, 211
  hpss_Fopen, 211
  hpss_Fread, 212
  hpss_Fseek, 214
  hpss_Fsync, 215
  hpss_Ftell, 215
  hpss_Fwrite, 215

CFG_COMPOUND_TYPE
  HPSS Config File API, 322
CFG_FLAG_TYPE
  HPSS Config File API, 322
CFG_SIMPLE_TYPE
  HPSS Config File API, 322
COSId
  hpss_Attrs, 352
CS_DEC64_LEN
  u_signed64.h, 566
cast32m
  64-bit Math Library, 264
cast64m
  64-bit Math Library, 264
chkbit64m
  64-bit Math Library, 264
Client API
  HPSS_PIO_MVR_SELECT, 83
  HPSS_PIO_READ, 83
  HPSS_PIO_TCPIP, 83
  HPSS_PIO_WRITE, 83
  HPSS_UNDELETE_NONE, 83
  HPSS_UNDELETE_OVERWRITE, 83
  HPSS_UNDELETE_OVERWRITE_AND_RESTORE, 83
  HPSS_UNDELETE_RESTORE_TIME, 83
  NAMESPEC_GROUP, 84
  NAMESPEC_REALM, 84
  NAMESPEC_SKIP, 84
  NAMESPEC_USER, 84
  NOTRUNC_CLEAR, 84
  NOTRUNC_SET, 84
  PURGE_LOCK, 84
  PURGE_ON_MIGRATE_CLEAR, 84
  PURGE_ON_MIGRATE_SET, 84
  PURGE_UNLOCK, 84
  SUPER_PURGE_LOCK, 84
  SUPER_PURGE_UNLOCK, 84
Client API, 65
  API_AddAllRegisterValues, 85
  API_AddRegisterValues, 85
  API_CONFIG_DEFAULT, 79
  API_DEBUG_ERROR, 79
  API_DEBUG_NONE, 79
  API_DEBUG_REQUEST, 79
  API_DEBUG_TRACE, 79
  API_ENABLE_LOGGING, 79
  API_GetServerIDForSubsystem, 85
  API_GetUniqueRequestID, 86
  API_MSGTYPE_PDATA, 80
  API_RemoveRegisterValues, 86
  API_StageBatchGetEntry, 86
  API_StageBatchInsertBFHandle, 86
  API_StageBatchInsertBFObj, 87
  API_StageBatchInsertFd, 88
  API_TRANSFER_TCP, 80
  API_USE_CONFIG, 80
  API_USE_SAN3P, 80
  api_namespec_t, 82
  HPSS_FILE, 82
  HPSS_MIGRATE_FORCE, 80
  HPSS_O_EXCL, 80
  HPSS_O_NO_TAPE, 81
  HPSS_O_STAGE_NONE, 81
  HPSS_PIO_PUSH, 81
  hpss_AbortIOByMoverId, 88
  hpss_AbortIOByRequestId, 89
  hpss_AbortIOByRequestMatch, 89
  hpss_AbortIOByUserId, 90
  hpss_AcctCodeToName, 90
  hpss_AcctNameToCode, 91
  hpss_BuildLevelString, 91
  hpss_ClearLastHPSSErrno, 91
  hpss_ConvertIdsToNames, 91
  hpss_ConvertNamesToIds, 92
  hpss_CopyrightStrings, 93
  hpss_FdelFileDigestList, 93
  hpss_FgetFileDigest, 94
  hpss_FgetFileDigestList, 94
  hpss_FileGetDigestHandle, 95
  hpss_FileGetDigestListHandle, 96
  hpss_FsetFileDigest, 96
  hpss_FsetFileDigestList, 97
  hpss_GetAcct, 97
  hpss_GetAcctName, 98
  hpss_GetAllSubsystems, 98
  hpss_GetAsyncStatus, 98
  hpss_GetBatchAsynchStatus, 99
  hpss_GetDistFile, 99

hpss_GetFullPath, 100
hpss_GetFullPathBitfile, 100
hpss_GetFullPathBitfileLen, 101
hpss_GetFullPathHandle, 101
hpss_GetFullPathHandleLen, 101
hpss_GetFullPathLen, 102
hpss_GetGID, 102
hpss_GetLastHPSSErrno, 102
hpss_GetNextIORequestID, 103
hpss_HashFlagsToString, 103
hpss_InsertDistFile, 103
hpss_LoadThreadState, 104
hpss_LookupRootCS, 104
hpss_Migrate, 104
hpss_PingCore, 105
hpss_Purge, 105
hpss_SetAcct, 106
hpss_SetAcctByName, 106
hpss_SetGID, 107
hpss_SiteIdToName, 107
hpss_SiteNameToId, 107
hpss_Stage, 108
hpss_StageBatch, 108
hpss_StageBatchCallBack, 109
hpss_StageCallBack, 109
hpss_StageCallBackBitfile, 110
hpss_Umask, 111
hpss_UserAttrListAttrHandle, 111
hpss_errno_state_t, 82
hpss_pio_operation_t, 83
hpss_pio_options_t, 83
hpss_pio_transport_t, 83
hpss_undelete_flags_t, 83
hpss_userattr_t, 83
MAX_XPATH_SIZE, 82
namespec_type_t, 83
notrunc_flag_t, 84
purgelock_flag_t, 84
purgeonmigrate_flag_t, 84
UDA_API_VALUE, 82
UDA_API_XML, 82
XML_ATTR, 82
XML_NO_ATTR, 82
Client API Control, 192
API_GetDataThreadStackSize, 192
API_SocketSetOptions, 192
hpss_ClientAPIInit, 194
hpss_ClientAPIReset, 194
hpss_GetConfiguration, 194
hpss_LoadDefaultThreadState, 194
hpss_MiscAdmin, 195
hpss_SetAPILogLevel, 196
hpss_SetAPILogPath, 196
hpss_SetApplicationName, 197
hpss_SetConfiguration, 197
hpss_ThreadCleanUp, 197
clrbit64m
64-bit Math Library, 264
Comment
hpss_Attrs, 352
Common_hpss_Rmdir
api_rmdir.c, 437
Common_hpss_RmdirHandle
api_rmdir.c, 437
Common_hpss_Unlink
api_unlink.c, 447
Common_hpss_UnlinkHandle
File Open, Create, and Close, 114
CompositePerms
hpss_Attrs, 352
core_api_def.x
HPSS_STAGE_STATUS_BFS, 461
HPSS_STAGE_STATUS_BFS_SS, 461
core_api_def.x, 457
hpss_stage_status_type_t, 460

DEFAULT_HASHTABLE_SIZE
Data Containers, 293
Data Containers, 287
DEFAULT_HASHTABLE_SIZE, 293
hpss_ihashtable_constants, 293
hpss_ilist_alloc_func_t, 291
hpss_ilist_append, 293
hpss_ilist_clear, 294
hpss_ilist_custom_init, 294
hpss_ilist_destructor_t, 291
hpss_ilist_empty, 294
hpss_ilist_foreach, 296
hpss_ilist_foreach_range, 296
hpss_ilist_get_current, 296
hpss_ilist_get_current_r, 297
hpss_ilist_get_node, 297
hpss_ilist_get_size, 297
hpss_ilist_halt_iterator, 297
hpss_ilist_init, 298
hpss_ilist_init_iterator, 298
hpss_ilist_init_iterator_r, 298
hpss_ilist_init_iterator_range, 299
hpss_ilist_init_iterator_range_r, 299
hpss_ilist_insert, 299
hpss_ilist_iterator_token_h, 291
hpss_ilist_join, 300
hpss_ilist_mapped_func_t, 292
hpss_ilist_prepend, 300
hpss_ilist_reinit_iterator, 300
hpss_ilist_reinit_iterator_range, 301
hpss_ilist_remove, 301
hpss_ilist_t, 292
hpss_irbtree_clear, 301
hpss_irbtree_color, 293
hpss_irbtree_container, 291
hpss_irbtree_empty, 302
hpss_irbtree_find, 302
hpss_irbtree_find_cmp, 302
hpss_irbtree_foreach, 303
hpss_irbtree_foreach_node, 303
hpss_irbtree_foreach_node_prev, 304

hpss_irbtree_init, 304
hpss_irbtree_insert, 304
hpss_irbtree_insert_multi, 305
hpss_irbtree_lower_bound, 305
hpss_irbtree_max, 305
hpss_irbtree_min, 305
hpss_irbtree_node_callback_t, 292
hpss_irbtree_node_comparator_t, 292
hpss_irbtree_node_destructor_t, 292
hpss_irbtree_node_next, 306
hpss_irbtree_node_prev, 306
hpss_irbtree_node_printer_t, 293
hpss_irbtree_print, 306
hpss_irbtree_remove, 307
hpss_irbtree_size, 307
hpss_irbtree_upper_bound, 307
IRBTREE_BLACK, 293
IRBTREE_RED, 293
MAX_LOAD_FACTOR, 293
Data Structure Conversion, 229
API_CalcBlock32, 229
API_CalcBlock64, 229
API_ConvertModeToPosixMode, 230
API_ConvertPosixModeToMode, 230
API_ConvertPosixTimeToTime, 230
API_ConvertTimeToPosixTime, 231
DataLength
hpss_Attrs, 352
dec64m
64-bit Math Library, 265
DirEntryTag, 558
Directory Creation and Deletion, 176
hpss_Mkdir, 176
hpss_MkdirHandle, 177
hpss_Rmdir, 177
hpss_RmdirHandle, 178
hpss_RmdirHandleImmediate, 178
hpss_RmdirImmediate, 179
Directory Data, 180
hpss_Closedir, 181
hpss_Getdents, 182
hpss_GetdentsAttrs, 182
hpss_Opendir, 183
hpss_OpendirHandle, 183
hpss_ReadAttrs, 184
hpss_ReadAttrsHandle, 184
hpss_ReadAttrsPlus, 185
hpss_ReadRawAttrsHandle, 187
hpss_Readdir, 185
hpss_ReaddirHandle, 186
hpss_ReaddirPlus, 186
hpss_Rewinddir, 188
div2x64m
64-bit Math Library, 265
div64_3m
64-bit Math Library, 265
div64m
64-bit Math Library, 265
DriveFlagStrings
hpss_String.c, 538
DriveStateStrings
hpss_String.c, 538

EntryCount
hpss_Attrs, 352
eq64m
64-bit Math Library, 265
eqz64m
64-bit Math Library, 265
ExtendedACLs
hpss_Attrs, 353

FamilyId
hpss_Attrs, 353
File Attribute, 145
hpss_Access, 146
hpss_AccessHandle, 147
hpss_Chacct, 147
hpss_ChacctByName, 148
hpss_Chmod, 148
hpss_Chown, 149
hpss_FileGetAttributes, 149
hpss_FileGetAttributesBitfile, 150
hpss_FileGetAttributesHandle, 150
hpss_FileGetXAttributes, 151
hpss_FileGetXAttributesHandle, 152
hpss_FileSetAttributes, 153
hpss_FileSetAttributesBitfile, 155
hpss_FileSetAttributesHandle, 156
hpss_FileSetCOS, 156
hpss_FileSetCOSHandle, 157
hpss_Fstat, 157
hpss_GetAttrHandle, 158
hpss_GetFileNotrunc, 159
hpss_GetListAttrs, 159
hpss_GetRawAttrHandle, 159
hpss_Lstat, 160
hpss_PurgeLock, 160
hpss_PurgeOnMigrate, 161
hpss_SetAttrHandle, 161
hpss_SetCOSByHints, 162
hpss_SetFileNotrunc, 163
hpss_Stat, 163
hpss_Utime, 164
File Data, 125
hpss_CopyFile, 126
hpss_Fclear, 126
hpss_FclearOffset, 127
hpss_Fpreallocate, 127
hpss_Ftruncate, 127
hpss_Lseek, 128
hpss_PreferredReadOrder, 128
hpss_Read, 129
hpss_ReadItem, 129
hpss_ReadQueueAdd, 131
hpss_ReadQueueAddMany, 131
hpss_ReadQueueCreateContext, 132

hpss_ReadQueueList, 132
hpss_ReadQueueRemove, 132
hpss_ReadQueueRemoveContext, 133
hpss_SetFileOffset, 133
hpss_SetNextIORequestID, 134
hpss_Truncate, 134
hpss_TruncateHandle, 134
hpss_Write, 135
File hash flag values, 63
File Name, 165
hpss_Link, 165
hpss_LinkHandle, 166
hpss_LinkHandleParent, 166
hpss_Rename, 167
hpss_RenameHandle, 168
hpss_Symlink, 168
hpss_SymlinkHandle, 169
File Open, Create, and Close, 113
Common_hpss_UnlinkHandle, 114
hpss_Close, 114
hpss_Creat, 114
hpss_Create, 115
hpss_CreateHandle, 116
hpss_Open, 116
hpss_OpenBitfile, 117
hpss_OpenBitfileVAttrs, 118
hpss_OpenHandle, 119
hpss_ReopenBitfile, 121
hpss_Unlink, 122
hpss_UnlinkHandle, 122
hpss_UnlinkHandleImmediate, 123
hpss_UnlinkImmediate, 124
Fileset and Junction, 137
hpss_FilesetCreate, 137
hpss_FilesetDelete, 138
hpss_FilesetGetAttributes, 139
hpss_FilesetListAll, 139
hpss_FilesetSetAttributes, 140
hpss_GetJunctionAttrs, 140
hpss_GetJunctions, 141
hpss_JunctionCreate, 141
hpss_JunctionCreateHandle, 142
hpss_JunctionDelete, 142
hpss_JunctionDeleteHandle, 144
FilesetHandle
hpss_Attrs, 353
FilesetId
hpss_Attrs, 353
FilesetRootObjectId
hpss_Attrs, 353
FilesetStateFlags
hpss_Attrs, 353
FilesetType
hpss_Attrs, 353
fsrootkeycompare
api_cache.c, 368

GK_CREATE_REQUEST
gk_gk_IFStructs.x, 462
GK_ERROR
gk_gk_IFStructs.x, 462
GK_GO
gk_gk_IFStructs.x, 462
GK_INVALID_REQUEST_TYPE
gk_gk_IFStructs.x, 462
GK_INVALID_STATE
gk_gk_IFStructs.x, 462
GK_OPEN_REQUEST
gk_gk_IFStructs.x, 462
GK_RETRY
gk_gk_IFStructs.x, 462
GK_SITE_CLOSE
gk_gk_IFStructs.x, 462
GK_SITE_CREATE
gk_gk_IFStructs.x, 462
GK_SITE_CREATE_COMPLETE
gk_gk_IFStructs.x, 462
GK_SITE_CREATE_STATS
gk_gk_IFStructs.x, 462
GK_SITE_OPEN
gk_gk_IFStructs.x, 462
GK_SITE_OPEN_STATS
gk_gk_IFStructs.x, 462
GK_SITE_STAGE
gk_gk_IFStructs.x, 462
GK_SITE_STAGE_COMPLETE
gk_gk_IFStructs.x, 462
GK_SITE_STAGE_STATS
gk_gk_IFStructs.x, 462
GK_STAGE_REQUEST
gk_gk_IFStructs.x, 462
GID
hpss_Attrs, 354
Gatekeeper Site Library, 334
gk_EntryInfo_t, 335
gk_ReqInfo_t, 336
gk_site_Close, 336
gk_site_CreateComplete, 336
gk_site_CreateStats, 337
gk_site_GetMonitorTypes, 338
gk_site_Init, 339
gk_site_Open, 339
gk_site_OpenStats, 340
gk_site_Stage, 340
gk_site_StageComplete, 341
gk_site_StageStats, 341
ge64m
64-bit Math Library, 265
gk_EntryInfo, 335
gk_OpenRequest, 461
gk_ReqInfo, 335
gk_gk_IFStructs.x
GK_CREATE_REQUEST, 462
GK_ERROR, 462
GK_GO, 462
GK_INVALID_REQUEST_TYPE, 462
GK_INVALID_STATE, 462

GK_OPEN_REQUEST, 462
GK_RETRY, 462
GK_SITE_CLOSE, 462
GK_SITE_CREATE, 462
GK_SITE_CREATE_COMPLETE, 462
GK_SITE_CREATE_STATS, 462
GK_SITE_OPEN, 462
GK_SITE_OPEN_STATS, 462
GK_SITE_STAGE, 462
GK_SITE_STAGE_COMPLETE, 462
GK_SITE_STAGE_STATS, 462
GK_STAGE_REQUEST, 462
gk_EntryInfo_t
Gatekeeper Site Library, 335
gk_RTM.h, 462
gk_ReqInfo_t
Gatekeeper Site Library, 336
gk_SiteProtos.h, 463
gk_State
gk_gk_IFStructs.x, 462
gk_gk_IFStructs.x, 461
gk_State, 462
gk_gk_RequestType, 462
gk_gk_RequestType
gk_gk_IFStructs.x, 462
gk_site_Close
Gatekeeper Site Library, 336
gk_site_CreateComplete
Gatekeeper Site Library, 336
gk_site_CreateStats
Gatekeeper Site Library, 337
gk_site_GetMonitorTypes
Gatekeeper Site Library, 338
gk_site_Init
Gatekeeper Site Library, 339
gk_site_Open
Gatekeeper Site Library, 339
gk_site_OpenStats
Gatekeeper Site Library, 340
gk_site_Stage
Gatekeeper Site Library, 340
gk_site_StageComplete
Gatekeeper Site Library, 341
gk_site_StageStats
Gatekeeper Site Library, 341
GroupPerms
hpss_Attrs, 354
gt64m
64-bit Math Library, 265

HPSS Config File API
CFG_COMPOUND_TYPE, 322
CFG_FLAG_TYPE, 322
CFG_SIMPLE_TYPE, 322
HPSS_FILE_HASH_DIGEST_ACTIVE
hpss_api.h, 482
HPSS_FILE_HASH_DIGEST_VALID
hpss_api.h, 482
HPSS_FILE_HASH_GENERATED
hpss_api.h, 482
HPSS_FILE_HASH_MIGRATE_VERIFIED
hpss_api.h, 482
HPSS_FILE_HASH_SET_BY_USER
hpss_api.h, 482
HPSS_FILE_HASH_SKIP_VERIFICATION
hpss_api.h, 482
HPSS_HASH_SINGLE
hpss_api.h, 482
HPSS_HASH_SINGLE_ONLY
hpss_api.h, 482
HPSS_HASH_STRIPE
hpss_api.h, 482
HPSS_HASH_STRIPE_ONLY
hpss_api.h, 482
HPSS_PIO_MVR_SELECT
Client API, 83
HPSS_PIO_READ
Client API, 83
HPSS_PIO_TCPIP
Client API, 83
HPSS_PIO_WRITE
Client API, 83
HPSS_READDIR_GETATTRS
hpss_api.h, 482
HPSS_READDIR_NFS
hpss_api.h, 482
HPSS_READDIR_NONE
hpss_api.h, 482
HPSS_READDIR_RAW
hpss_api.h, 482
HPSS_STAGE_STATUS_BFS
core_api_def.x, 461
HPSS_STAGE_STATUS_BFS_SS
core_api_def.x, 461
HPSS_UNDELETE_NONE
Client API, 83
HPSS_UNDELETE_OVERWRITE
Client API, 83
HPSS_UNDELETE_OVERWRITE_AND_RESTORE
Client API, 83
HPSS_UNDELETE_RESTORE_TIME
Client API, 83
HPSS Access Control List APIs, 326
hacl_ConvertACLToHACL, 328
hacl_ConvertHACLPermsToPerms, 328
hacl_ConvertHACLToACL, 329
hacl_ConvertHACLToString, 329
hacl_ConvertHACLTypeToType, 330
hacl_ConvertPermsToHACLPerms, 330
hacl_ConvertStringsToHACL, 331
hacl_ConvertTypeToHACLType, 331
hacl_DeleteHACL, 332
hacl_GetHACL, 332
hacl_SetHACL, 332
hacl_SortHACL, 333
hacl_UpdateHACL, 333
hacl_acl_entry_t, 328

HPSS Client API Extensions, 269
    hpsscfgx_ConfGetClientInterfaces, 270
    hpsscfgx_ConfParse, 271
    hpsscfgx_ConfSetDirPaths, 271
    hpsscfgx_ConfSetFileName, 272
    hpsscfgx_LookupHostName, 272
    hpsscfgx_NetoptFindEntry, 273
    hpsscfgx_NetoptGetWriteSize, 273
    hpsscfgx_NetoptSetSock, 273
    hpsscfgx_PatternMatch, 274
HPSS Config File API, 319
    hpss_CfgFindKey, 322
    hpss_CfgFindToken, 324
    hpss_CfgFree, 324
    hpss_CfgParse, 324
    hpss_cfg_stanza_t, 322
    hpsscfgx_ConfParse, 325
HPSS Config File Template API, 275
HPSS Environment Variable Module, 243
    hpss_EnvLocalUpdate, 243
    hpss_EnvReInitialize, 244
    hpss_Envcpy, 243
    hpss_Getenv, 244
HPSS Networking Library, 245
    hpss_net_accept, 246
    hpss_net_addrmatch, 247
    hpss_net_bind, 247
    hpss_net_connect, 247
    hpss_net_generatemask, 249
    hpss_net_get_family_option, 249
    hpss_net_get_family_string, 249
    hpss_net_getinaddr, 250
    hpss_net_getnameinfo, 250
    hpss_net_getpeername, 251
    hpss_net_getport, 251
    hpss_net_getuniversaladdress, 251
    hpss_net_initaddr, 252
    hpss_net_listen, 252
    hpss_net_setport, 252
    hpss_net_socket, 253
    hpss_net_strerror, 253
    hpss_net_universaladdresstoport, 253
HPSS Object, 205
    hpss_GetObjFromHandle, 205
    hpss_GetObjType, 205
    hpss_GetPathHandle, 206
    hpss_GetPathHandleObjHash, 206
    hpss_GetPathObjHash, 206
HPSS Statistics, 201
    hpss_GetSubSysLimits, 201
    hpss_GetSubSysStats, 201
    hpss_ResetSubSysStats, 202
    hpss_Statfs, 202
    hpss_Statfs64, 202
    hpss_Statvfs, 203
    hpss_Statvfs64, 203
HPSS XML Interface, 255
    hpss_ChompXMLHeader, 255
    hpss_CreateXMLWhere, 256
    hpss_CreateXQueryDelete, 256
    hpss_CreateXQueryGet, 256
    hpss_CreateXQueryUpdate, 257
    hpss_ValidXPath, 258
    hpss_XMLtoAttr, 258
    hpss_XPathBranch, 259
    hpss_XPathLeaf, 259
    hpss_XPathRBranch, 260
    hpss_XPathRoot, 260
    hpss_XPathtoXML, 261
HPSS_CKSUM_AFALG
    Hash Interface, 311
HPSS_CKSUM_SSE42
    Hash Interface, 311
HPSS_CKSUM_VPMSUM
    Hash Interface, 311
HPSS_EHACL_UNAUTH
    HPSS Access Control List APIs, 327
HPSS_FILE
    Client API, 82
HPSS_MIGRATE_FORCE
    Client API, 80
HPSS_O_EXCL
    Client API, 80
HPSS_O_NO_TAPE
    Client API, 81
HPSS_O_STAGE_ASYNC
    Client API, 81
HPSS_O_STAGE_BKGRD
    Client API, 81
HPSS_O_STAGE_NONE
    Client API, 81
HPSS_OFF_t
    hpss_types.x, 547
HPSS_PIO_PUSH
    Client API, 81
HPSSCFGX_ENUM
    Parsers, 279
HPSSCFGX_STRING
    Parsers, 279
hacl.h, 463
hacl_acl_entry, 327
hacl_ConvertACLToHACL
    HPSS Access Control List APIs, 328
hacl_ConvertHACLPermsToPerms
    HPSS Access Control List APIs, 328
hacl_ConvertHACLToACL
    HPSS Access Control List APIs, 329
hacl_ConvertHACLToString
    HPSS Access Control List APIs, 329
hacl_ConvertHACLTypeToType
    HPSS Access Control List APIs, 330
hacl_ConvertPermsToHACLPerms
    HPSS Access Control List APIs, 330
hacl_ConvertStringsToHACL
    HPSS Access Control List APIs, 331
hacl_ConvertTypeToHACLType

HPSS Access Control List APIs, 331
hacl_DeleteHACL
HPSS Access Control List APIs, 332
hacl_GetHACL
HPSS Access Control List APIs, 332
hacl_SetHACL
HPSS Access Control List APIs, 332
hacl_SortHACL
HPSS Access Control List APIs, 333
hacl_UpdateHACL
HPSS Access Control List APIs, 333
hacl_acl_entry_t
HPSS Access Control List APIs, 328
Hash Interface, 310
HPSS_CKSUM_AFALG, 311
HPSS_CKSUM_SSE42, 311
HPSS_CKSUM_VPMSUM, 311
hpss_HashAppend, 311
hpss_HashAppendBuf, 311
hpss_HashAppendStr, 312
hpss_HashBaseInit, 312
hpss_HashCreate, 312
hpss_HashDecode, 313
hpss_HashDelete, 313
hpss_HashDigestLength, 313
hpss_HashDigestToHex, 314
hpss_HashDigestToString, 314
hpss_HashDuplicate, 314
hpss_HashEncode, 315
hpss_HashExtract, 315
hpss_HashFinish, 315
hpss_HashFinishDigest, 316
hpss_HashFinishHex, 316
hpss_HashGetHWFlags, 316
hpss_HashLoad, 317
hpss_HashReset, 317
hpss_HashType, 317
hpss_HexStringToHashDigest, 318
high32m
64-bit Math Library, 266
hpss_TrashRecord, 544
hpss_api.h
HPSS_FILE_HASH_DIGEST_ACTIVE, 482
HPSS_FILE_HASH_DIGEST_VALID, 482
HPSS_FILE_HASH_GENERATED, 482
HPSS_FILE_HASH_MIGRATE_VERIFIED, 482
HPSS_FILE_HASH_SET_BY_USER, 482
HPSS_FILE_HASH_SKIP_VERIFICATION, 482
HPSS_HASH_SINGLE, 482
HPSS_HASH_SINGLE_ONLY, 482
HPSS_HASH_STRIPE, 482
HPSS_HASH_STRIPE_ONLY, 482
HPSS_READDIR_GETATTRS, 482
HPSS_READDIR_NFS, 482
HPSS_READDIR_NONE, 482
HPSS_READDIR_RAW, 482
hpss_authn_mech_gsi
hpss_types.x, 548
hpss_authn_mech_invalid
hpss_types.x, 548
hpss_authn_mech_krb5
hpss_types.x, 548
hpss_authn_mech_spkm
hpss_types.x, 548
hpss_authn_mech_unix
hpss_types.x, 548
hpss_authz_mech_invalid
hpss_types.x, 548
hpss_authz_mech_ldap
hpss_types.x, 548
hpss_authz_mech_unix
hpss_types.x, 548
hpss_cfg_stanza, 321
hpss_core_read, 458
hpss_core_stage, 458
hpss_core_stage_callback, 458
hpss_cos_hints, 491
hpss_cos_md, 491
hpss_cos_priorities, 493
hpss_cursor_id_t, 546
hpss_dirent, 495
hpss_errno_state, 74
hpss_file_hash_digest, 480
hpss_file_hash_digest_list, 481
hpss_fileattr, 490
hpss_global_fsent_t, 74
hpss_group_ids, 517
hpss_io_req_state_active
hpss_types.x, 549
hpss_io_req_state_complete
hpss_types.x, 549
hpss_io_req_state_defunct
hpss_types.x, 549
hpss_io_req_state_queued
hpss_types.x, 549
hpss_io_req_state_ready
hpss_types.x, 549
hpss_irbtree, 290
hpss_irbtree_node, 290
hpss_junction_ent_t, 74
hpss_migr_order_type_create_time
hpss_types.x, 549
hpss_migr_order_type_parent_id
hpss_types.x, 549
hpss_pio_gapinfo_s, 74
hpss_pio_params_s, 76
hpss_read_queue_item, 458
hpss_read_queue_list, 458
hpss_reqid_list, 459
hpss_rpc_auth_type_invalid
hpss_types.x, 549
hpss_rpc_auth_type_key
hpss_types.x, 549
hpss_rpc_auth_type_keyfile
hpss_types.x, 549
hpss_rpc_auth_type_keytab

hpss_types.x, 549
hpss_rpc_auth_type_none
hpss_types.x, 549
hpss_rpc_auth_type_passwd
hpss_types.x, 549
hpss_rpc_endpoint, 547
hpss_stage, 456
hpss_stage_batch, 457
hpss_stage_batch_status, 457
hpss_stage_bitfile_list, 457
hpss_stage_status, 457
hpss_stat, 76
hpss_statfs, 77
hpss_statfs64, 77
hpss_statvfs, 77
hpss_statvfs64, 78
hpss_string_list, 459
hpss_subsys_limits, 459
hpss_tape_schedule_fifo
hpss_types.x, 550
hpss_tape_schedule_offset
hpss_types.x, 550
hpss_tape_schedule_unknown
hpss_types.x, 550
hpss_types.x
hpss_authn_mech_gsi, 548
hpss_authn_mech_invalid, 548
hpss_authn_mech_krb5, 548
hpss_authn_mech_spkm, 548
hpss_authn_mech_unix, 548
hpss_authz_mech_invalid, 548
hpss_authz_mech_ldap, 548
hpss_authz_mech_unix, 548
hpss_io_req_state_active, 549
hpss_io_req_state_complete, 549
hpss_io_req_state_defunct, 549
hpss_io_req_state_queued, 549
hpss_io_req_state_ready, 549
hpss_migr_order_type_create_time, 549
hpss_migr_order_type_parent_id, 549
hpss_rpc_auth_type_invalid, 549
hpss_rpc_auth_type_key, 549
hpss_rpc_auth_type_keyfile, 549
hpss_rpc_auth_type_keytab, 549
hpss_rpc_auth_type_none, 549
hpss_rpc_auth_type_passwd, 549
hpss_tape_schedule_fifo, 550
hpss_tape_schedule_offset, 550
hpss_tape_schedule_unknown, 550
lbp_verify_error_block_guard, 550
lbp_verify_error_block_method, 550
lbp_verify_error_errno, 550
lbp_verify_error_none, 550
hpss_userattr, 78
hpss_userattr_list, 78
hpss_xfileattr, 490
hpss_AbortIOByMoverId
Client API, 88
hpss_AbortIOByRequestId
Client API, 89
hpss_AbortIOByRequestMatch
Client API, 89
hpss_AbortIOByUserId
Client API, 90
hpss_Access
File Attribute, 146
hpss_AccessHandle
File Attribute, 147
hpss_AcctCodeToName
Client API, 90
hpss_AcctNameToCode
Client API, 91
hpss_AdminStateString
hpss_String.c, 522
hpss_AllocStateString
hpss_String.c, 522
hpss_Attrs, 351
Account, 352
Activity, 352
BitfileObj, 352
COSId, 352
Comment, 352
CompositePerms, 352
DataLength, 352
EntryCount, 352
ExtendedACLs, 353
FamilyId, 353
FilesetHandle, 353
FilesetId, 353
FilesetRootObjectId, 353
FilesetStateFlags, 353
FilesetType, 353
GID, 354
GroupPerms, 354
LinkCount, 354
ModePerms, 354
OpenCount, 354
OptionFlags, 354
OtherPerms, 354
ReadCount, 354
RealmId, 354
RegisterBitMap, 354
SubSystemId, 354
TimeCreated, 355
TimeLastRead, 355
TimeLastWritten, 355
TimeModified, 355
TrashInfo, 355
Type, 355
UID, 355
UserPerms, 355
WriteCount, 355
hpss_AuthenticatorTypeFromString
hpss_String.c, 522
hpss_AuthenticatorTypeString
hpss_String.c, 522

hpss_AuthnMechTypeFromString
  hpss_String.c, 523
hpss_AuthnMechTypeString
  hpss_String.c, 523
hpss_AuthzMechTypeFromString
  hpss_String.c, 523
hpss_AuthzMechTypeString
  hpss_String.c, 524
hpss_BuildLevelString
  Client API, 91
hpss_Bytecount32
  hpss_String.c, 524
hpss_Bytecount64
  hpss_String.c, 524
hpss_CfgFindKey
  HPSS Config File API, 322
hpss_CfgFindToken
  HPSS Config File API, 324
hpss_CfgFree
  HPSS Config File API, 324
hpss_CfgParse
  HPSS Config File API, 324
hpss_Chacct
  File Attribute, 147
hpss_ChacctByName
  File Attribute, 148
hpss_Chdir
  Working Directory, 189
hpss_Chmod
  File Attribute, 148
hpss_ChompXMLHeader
  HPSS XML Interface, 255
hpss_Chown
  File Attribute, 149
hpss_Chroot
  Working Directory, 189
hpss_ClearLastHPSSErrno
  Client API, 91
hpss_ClientAPIInit
  Client API Control, 194
hpss_ClientAPIReset
  Client API Control, 194
hpss_Close
  File Open, Create, and Close, 114
hpss_Closedir
  Directory Data, 181
hpss_ConvertIdsToNames
  Client API, 91
hpss_ConvertNamesToIds
  Client API, 92
hpss_CopyFile
  File Data, 126
hpss_CopyrightStrings
  Client API, 93
hpss_Creat
  File Open, Create, and Close, 114
hpss_Create
  File Open, Create, and Close, 115
hpss_CreateHandle
  File Open, Create, and Close, 116
hpss_CreateWithHints
  POSIX APIs, 282
hpss_CreateXMLWhere
  HPSS XML Interface, 256
hpss_CreateXQueryDelete
  HPSS XML Interface, 256
hpss_CreateXQueryGet
  HPSS XML Interface, 256
hpss_CreateXQueryUpdate
  HPSS XML Interface, 257
hpss_DeleteACL
  ACLs, 170
hpss_DeleteACLHandle
  ACLs, 171
hpss_DriveFlagString
  hpss_String.c, 524
hpss_DriveStateString
  hpss_String.c, 525
hpss_DriveTypeString
  hpss_String.c, 525
hpss_EnvLocalUpdate
  HPSS Environment Variable Module, 243
hpss_EnvReInitialize
  HPSS Environment Variable Module, 244
hpss_Envcpy
  HPSS Environment Variable Module, 243
hpss_ErrnoName
  hpss_String.c, 525
hpss_Fclear
  File Data, 126
hpss_FclearOffset
  File Data, 127
hpss_Fclose
  Buffered Data, 208
hpss_Fcntl
  Buffered Data, 209
hpss_FdelFileDigestList
  Client API, 93
hpss_Fflush
  Buffered Data, 210
hpss_FgetFileDigest
  Client API, 94
hpss_FgetFileDigestList
  Client API, 94
hpss_Fgetc
  Buffered Data, 210
hpss_Fgets
  Buffered Data, 211
hpss_FileGetAttributes
  File Attribute, 149
hpss_FileGetAttributesBitfile
  File Attribute, 150
hpss_FileGetAttributesHandle
  File Attribute, 150
hpss_FileGetDigestHandle
  Client API, 95

hpss_FileGetDigestListHandle
Client API, 96
hpss_FileGetXAttributes
File Attribute, 151
hpss_FileGetXAttributesHandle
File Attribute, 152
hpss_FileSetAttributes
File Attribute, 153
hpss_FileSetAttributesBitfile
File Attribute, 155
hpss_FileSetAttributesHandle
File Attribute, 156
hpss_FileSetCOS
File Attribute, 156
hpss_FileSetCOSHandle
File Attribute, 157
hpss_FilesetCreate
Fileset and Junction, 137
hpss_FilesetDelete
Fileset and Junction, 138
hpss_FilesetGetAttributes
Fileset and Junction, 139
hpss_FilesetListAll
Fileset and Junction, 139
hpss_FilesetSetAttributes
Fileset and Junction, 140
hpss_Fopen
Buffered Data, 211
hpss_Fpreallocate
File Data, 127
hpss_Fread
Buffered Data, 212
hpss_Fseek
Buffered Data, 214
hpss_FsetFileDigest
Client API, 96
hpss_FsetFileDigestList
Client API, 97
hpss_Fstat
File Attribute, 157
hpss_Fsync
Buffered Data, 215
hpss_Ftell
Buffered Data, 215
hpss_Ftruncate
File Data, 127
hpss_Fwrite
Buffered Data, 215
hpss_GetACL
ACLs, 172
hpss_GetACLHandle
ACLs, 172
hpss_GetAcct
Client API, 97
hpss_GetAcctName
Client API, 98
hpss_GetAllSubsystems
Client API, 98
hpss_GetAsyncStatus
Client API, 98
hpss_GetAttrHandle
File Attribute, 158
hpss_GetAuthType
hpss_Mech.c, 509
hpss_GetBatchAsynchStatus
Client API, 99
hpss_GetConfiguration
Client API Control, 194
hpss_GetDistFile
Client API, 99
hpss_GetFileNotrunc
File Attribute, 159
hpss_GetFullPath
Client API, 100
hpss_GetFullPathBitfile
Client API, 100
hpss_GetFullPathBitfileLen
Client API, 101
hpss_GetFullPathHandle
Client API, 101
hpss_GetFullPathHandleLen
Client API, 101
hpss_GetFullPathLen
Client API, 102
hpss_GetGID
Client API, 102
hpss_GetIntersectedList
api_misc.c, 418
hpss_api.h, 485
hpss_GetJunctionAttrs
Fileset and Junction, 140
hpss_GetJunctions
Fileset and Junction, 141
hpss_GetLastHPSSErrno
Client API, 102
hpss_GetListAttrs
File Attribute, 159
hpss_GetNextIORequestID
Client API, 103
hpss_GetObjFromHandle
HPSS Object, 205
hpss_GetObjType
HPSS Object, 205
hpss_GetPathHandle
HPSS Object, 206
hpss_GetPathHandleObjHash
HPSS Object, 206
hpss_GetPathObjHash
HPSS Object, 206
hpss_GetRawAttrHandle
File Attribute, 159
hpss_GetSubSysLimits
HPSS Statistics, 201
hpss_GetSubSysStats
HPSS Statistics, 201
hpss_GetThreadUcred

Working Directory, 190
hpss_GetTrashSettings
Trashcan, 239
hpss_GetTrashcan
Trashcan, 238
hpss_GetTrashcanHandle
Trashcan, 239
hpss_GetUID
api_ids.c, 382
hpss_api.h, 485
hpss_Getcwd
Working Directory, 190
hpss_Getdents
Directory Data, 182
hpss_GetdentsAttrs
Directory Data, 182
hpss_Getenv
HPSS Environment Variable Module, 244
hpss_Getenv.c, 495
hpss_GetenvDefault, 496
hpss_Getenv.h, 496
hpss_GetenvDefault, 496
hpss_GetenvDefault
hpss_Getenv.c, 496
hpss_Getenv.h, 496
hpss_HashAppend
Hash Interface, 311
hpss_HashAppendBuf
Hash Interface, 311
hpss_HashAppendStr
Hash Interface, 312
hpss_HashBaseInit
Hash Interface, 312
hpss_HashCreate
Hash Interface, 312
hpss_HashDecode
Hash Interface, 313
hpss_HashDelete
Hash Interface, 313
hpss_HashDigestLength
Hash Interface, 313
hpss_HashDigestToHex
Hash Interface, 314
hpss_HashDigestToString
Hash Interface, 314
hpss_HashDuplicate
Hash Interface, 314
hpss_HashEncode
Hash Interface, 315
hpss_HashExtract
Hash Interface, 315
hpss_HashFinish
Hash Interface, 315
hpss_HashFinishDigest
Hash Interface, 316
hpss_HashFinishHex
Hash Interface, 316
hpss_HashFlagsToString
Client API, 103
hpss_HashGetHWFlags
Hash Interface, 316
hpss_HashLoad
Hash Interface, 317
hpss_HashReset
Hash Interface, 317
hpss_HashType
Hash Interface, 317
hpss_HashTypeString
hpss_String.c, 526
hpss_HexString
hpss_String.c, 526
hpss_HexStringToHashDigest
Hash Interface, 318
hpss_InsertDistFile
Client API, 103
hpss_JunctionCreate
Fileset and Junction, 141
hpss_JunctionCreateHandle
Fileset and Junction, 142
hpss_JunctionDelete
Fileset and Junction, 142
hpss_JunctionDeleteHandle
Fileset and Junction, 144
hpss_LabelTypeString
hpss_String.c, 526
hpss_Link
File Name, 165
hpss_LinkHandle
File Name, 166
hpss_LinkHandleParent
File Name, 166
hpss_LoadDefaultThreadState
Client API Control, 194
hpss_LoadThreadState
Client API, 104
hpss_LookupRootCS
Client API, 104
hpss_Lseek
File Data, 128
hpss_Lstat
File Attribute, 160
hpss_Mech.c, 508
hpss_GetAuthType, 509
hpss_MediaTypeString
hpss_String.c, 526
hpss_Migrate
Client API, 104
hpss_MiscAdmin
Client API Control, 195
hpss_Mkdir
Directory Creation and Deletion, 176
hpss_MkdirHandle
Directory Creation and Deletion, 177
hpss_MountModeString
hpss_String.c, 528
hpss_OpStateString

hpss_String.c, 528
hpss_Open
File Open, Create, and Close, 116
hpss_OpenBitfile
File Open, Create, and Close, 117
hpss_OpenBitfileVAttrs
File Open, Create, and Close, 118
hpss_OpenHandle
File Open, Create, and Close, 119
hpss_OpenWithHints
POSIX APIs, 283
hpss_Opendir
Directory Data, 183
hpss_OpendirHandle
Directory Data, 183
hpss_PIOEnd
Parallel I/O, 232
hpss_PIOExecute
Parallel I/O, 233
hpss_PIOExecuteItem
Parallel I/O, 233
hpss_PIOExportGrp
Parallel I/O, 234
hpss_PIOFinalize
Parallel I/O, 234
hpss_PIOGetPartError
Parallel I/O, 235
hpss_PIOGetRequestID
Parallel I/O, 235
hpss_PIOImportGrp
Parallel I/O, 235
hpss_PIORegister
Parallel I/O, 236
hpss_PIOStart
Parallel I/O, 236
hpss_PVLActivityStateString
hpss_String.c, 529
hpss_PVLJobStateString
hpss_String.c, 529
hpss_PVRTypeString
hpss_String.c, 529
hpss_PingCore
Client API, 105
hpss_PosixCreate
POSIX APIs, 283
hpss_PosixFstat
POSIX APIs, 283
hpss_PosixLseek
POSIX APIs, 284
hpss_PosixLstat
POSIX APIs, 284
hpss_PosixOpen
POSIX APIs, 284
hpss_PosixStat
POSIX APIs, 286
hpss_PreferredReadOrder
File Data, 128
hpss_ProtLevelFromString
hpss_String.c, 528
hpss_ProtLevelString
hpss_String.c, 528
hpss_Purge
Client API, 105
hpss_PurgeLock
File Attribute, 160
hpss_PurgeLoginCred
Authentication, 198
hpss_PurgeOnMigrate
File Attribute, 161
hpss_Read
File Data, 129
hpss_ReadAttrs
Directory Data, 184
hpss_ReadAttrsHandle
Directory Data, 184
hpss_ReadAttrsPlus
Directory Data, 185
hpss_ReadItem
File Data, 129
hpss_ReadList
api_rdlist.c, 433
hpss_api.h, 487
hpss_ReadQueueAdd
File Data, 131
hpss_ReadQueueAddMany
File Data, 131
hpss_ReadQueueCreateContext
File Data, 132
hpss_ReadQueueList
File Data, 132
hpss_ReadQueueRemove
File Data, 132
hpss_ReadQueueRemoveContext
File Data, 133
hpss_ReadRawAttrsHandle
Directory Data, 187
hpss_ReadTrash
Trashcan, 240
hpss_Readdir
Directory Data, 185
hpss_ReaddirHandle
Directory Data, 186
hpss_ReaddirPlus
Directory Data, 186
hpss_Readlink
api_rdlink.c, 432
hpss_api.h, 486
hpss_ReadlinkHandle
api_rdlink.c, 432
hpss_api.h, 486
hpss_RemoveIntersectionFromList
api_misc.c, 418
hpss_api.h, 487
hpss_Rename
File Name, 167
hpss_RenameHandle

File Name, 168
hpss_ReopenBitfile
File Open, Create, and Close, 121
hpss_ResetSubSysStats
HPSS Statistics, 202
hpss_Rewinddir
Directory Data, 188
hpss_Rmdir
Directory Creation and Deletion, 177
hpss_RmdirHandle
Directory Creation and Deletion, 178
hpss_RmdirHandleImmediate
Directory Creation and Deletion, 178
hpss_RmdirImmediate
Directory Creation and Deletion, 179
hpss_Rumble
api_rddir.c, 431
hpss_api.h, 487
hpss_ServerTypeString
hpss_String.c, 529
hpss_SetACL
ACLs, 173
hpss_SetACLHandle
ACLs, 174
hpss_SetAPILogLevel
Client API Control, 196
hpss_SetAPILogPath
Client API Control, 196
hpss_SetAcct
Client API, 106
hpss_SetAcctByName
Client API, 106
hpss_SetAppLoginCred
Authentication, 198
hpss_SetApplicationName
Client API Control, 197
hpss_SetAttrHandle
File Attribute, 161
hpss_SetAuditInfo
Authentication, 199
hpss_SetCOSByHints
File Attribute, 162
hpss_SetConfiguration
Client API Control, 197
hpss_SetFileNotrunc
File Attribute, 163
hpss_SetFileOffset
File Data, 133
hpss_SetGID
Client API, 107
hpss_SetLoginCred
Authentication, 199
hpss_SetNextIORequestID
File Data, 134
hpss_SiteIdToName
Client API, 107
hpss_SiteNameToId
Client API, 107
hpss_Stage
Client API, 108
hpss_StageBatch
Client API, 108
hpss_StageBatchCallBack
Client API, 109
hpss_StageCallBack
Client API, 109
hpss_StageCallBackBitfile
Client API, 110
hpss_Stat
File Attribute, 163
hpss_Statfs
HPSS Statistics, 202
hpss_Statfs64
HPSS Statistics, 202
hpss_Statvfs
HPSS Statistics, 203
hpss_Statvfs64
HPSS Statistics, 203
hpss_String.c, 518
AdminStateStrings, 537
AuthenticatorTypeStrings, 537
AuthnMechStrings, 538
AuthzMechStrings, 538
DriveFlagStrings, 538
DriveStateStrings, 538
hpss_AdminStateString, 522
hpss_AllocStateString, 522
hpss_AuthenticatorTypeFromString, 522
hpss_AuthenticatorTypeString, 522
hpss_AuthnMechTypeFromString, 523
hpss_AuthnMechTypeString, 523
hpss_AuthzMechTypeFromString, 523
hpss_AuthzMechTypeString, 524
hpss_Bytecount32, 524
hpss_Bytecount64, 524
hpss_DriveFlagString, 524
hpss_DriveStateString, 525
hpss_DriveTypeString, 525
hpss_ErrnoName, 525
hpss_HashTypeString, 526
hpss_HexString, 526
hpss_LabelTypeString, 526
hpss_MediaTypeString, 526
hpss_MountModeString, 528
hpss_OpStateString, 528
hpss_PVLActivityStateString, 529
hpss_PVLJobStateString, 529
hpss_PVRTypeString, 529
hpss_ProtLevelFromString, 528
hpss_ProtLevelString, 528
hpss_ServerTypeString, 529
hpss_StringClean, 533
hpss_StringsAreEqual, 535
hpss_Token_Add, 535
hpss_Tokenize, 536
hpss_UsageStateString, 536

hpss_getline, 525
hpss_string_cmp, 530
hpss_string_fromCstr, 530
hpss_stringbuf_clear, 530
hpss_stringbuf_cmp, 530
hpss_stringbuf_delBackward, 531
hpss_stringbuf_delForward, 531
hpss_stringbuf_free, 531
hpss_stringbuf_fromCstr, 532
hpss_stringbuf_fromFile, 532
hpss_stringbuf_fromString, 532
hpss_stringbuf_grow, 532
hpss_stringbuf_insert, 533
hpss_stringbuf_printf, 533
hpss_time_string, 535
LabelTypeStrings, 539
NumAdminStateStrings, 539
NumAuthenticatorTypeStrings, 539
NumAuthnMechStrings, 539
NumAuthzMechStrings, 539
NumDriveFlagStrings, 540
NumDriveStateStrings, 540
NumErrnoStrings, 540
NumLabelTypeStrings, 540
NumOpStateStrings, 540
NumPVLActivityStateStrings, 541
NumPVLJobStateStrings, 541
NumPVRTypeStrings, 541
NumProtLevelStrings, 540
NumServerSubtypeStrings, 541
NumServerTypeStrings, 541
NumUsageStateStrings, 541
OpStateStrings, 542
PVLActivityStateStrings, 542
PVLJobStateStrings, 542
PVRTypeStrings, 543
ProtLevelStrings, 542
s32_and_map, 536
s32_eq_map, 536
ServerSubtypeStrings, 543
ServerTypeStrings, 543
u32_and_map, 537
u32_eq_map, 537
UsageStateStrings, 544
hpss_StringClean
hpss_String.c, 533
hpss_StringsAreEqual
hpss_String.c, 535
hpss_Symlink
File Name, 168
hpss_SymlinkHandle
File Name, 169
hpss_ThreadCleanUp
Client API Control, 197
hpss_Token_Add
hpss_String.c, 535
hpss_Tokenize
hpss_String.c, 536
hpss_TrashRecord.x, 544
hpss_Truncate
File Data, 134
hpss_TruncateHandle
File Data, 134
hpss_Umask
Client API, 111
hpss_Undelete
Trashcan, 240
hpss_UndeleteHandle
Trashcan, 241
hpss_Unlink
File Open, Create, and Close, 122
hpss_UnlinkHandle
File Open, Create, and Close, 122
hpss_UnlinkHandleImmediate
File Open, Create, and Close, 123
hpss_UnlinkImmediate
File Open, Create, and Close, 124
hpss_UpdateACL
ACLs, 174
hpss_UpdateACLHandle
ACLs, 175
hpss_UsageStateString
hpss_String.c, 536
hpss_UserAttrCloseCursor
User-defined Attributes, 218
hpss_UserAttrDeleteAttrHandle
User-defined Attributes, 218
hpss_UserAttrDeleteAttrs
User-defined Attributes, 219
hpss_UserAttrGetAttrHandle
User-defined Attributes, 219
hpss_UserAttrGetAttrs
User-defined Attributes, 220
hpss_UserAttrGetObj
User-defined Attributes, 220
hpss_UserAttrGetObjQuery
User-defined Attributes, 221
hpss_UserAttrGetXML
User-defined Attributes, 221
hpss_UserAttrGetXMLObj
User-defined Attributes, 222
hpss_UserAttrListAttrHandle
Client API, 111
hpss_UserAttrListAttrs
User-defined Attributes, 223
hpss_UserAttrReadObj
User-defined Attributes, 223
hpss_UserAttrReadXML
User-defined Attributes, 224
hpss_UserAttrReadXMLObj
User-defined Attributes, 224
hpss_UserAttrSetAttrHandle
User-defined Attributes, 225
hpss_UserAttrSetAttrs
User-defined Attributes, 225
hpss_UserAttrXQueryGet

User-defined Attributes, 226
hpss_UserAttrXQueryGetHandle
User-defined Attributes, 226
hpss_UserAttrXQueryUpdate
User-defined Attributes, 227
hpss_UserAttrXQueryUpdateHandle
User-defined Attributes, 227
hpss_Utime
File Attribute, 164
hpss_ValidXPath
HPSS XML Interface, 258
hpss_Write
File Data, 135
hpss_WriteList
api_wrtlist.c, 452
hpss_api.h, 489
hpss_XMLtoAttr
HPSS XML Interface, 258
hpss_XPathBranch
HPSS XML Interface, 259
hpss_XPathLeaf
HPSS XML Interface, 259
hpss_XPathRBranch
HPSS XML Interface, 260
hpss_XPathRoot
HPSS XML Interface, 260
hpss_XPathtoXML
HPSS XML Interface, 261
hpss_api.h, 465
API_FreeIOR, 482
API_GetBatchStatus, 483
API_GetClientAddr, 483
API_StageBatchFree, 483
API_StageBatchInit, 485
API_StageStatusFree, 485
hpss_GetIntersectedList, 485
hpss_GetUID, 485
hpss_ReadList, 487
hpss_Readlink, 486
hpss_ReadlinkHandle, 486
hpss_RemoveIntersectionFromList, 487
hpss_Rumble, 487
hpss_WriteList, 489
hpss_file_hash_flags, 481
hpss_file_hash_flags_t, 481
hpss_file_hash_stripe_flags, 482
hpss_readdir_flags_t, 482
hpss_api_def.x, 489
hpss_authn_mech_t
hpss_types.x, 548
hpss_authz_mech_t
hpss_types.x, 548
hpss_cfg_stanza_t
HPSS Config File API, 322
hpss_cos.x, 490
hpss_db_partition_t
hpss_types.x, 547
hpss_dirent.h, 494
hpss_distributionkey_t
hpss_types.x, 547
hpss_errno_state_t
Client API, 82
hpss_file_hash_flags
hpss_api.h, 481
hpss_file_hash_flags_t
hpss_api.h, 481
hpss_file_hash_stripe_flags
hpss_api.h, 482
hpss_getline
hpss_String.c, 525
hpss_hash.c, 497
ADLER32_BASE, 498
hpss_hash.h, 498
hpss_hash_enum
hpss_types.x, 548
hpss_hash_type_last
hpss_types.x, 550
hpss_id_t
hpss_types.x, 547
hpss_ihashtable.h, 500
hpss_ihashtable_node_t, 500
hpss_ihashtable_constants
Data Containers, 293
hpss_ihashtable_node_t
hpss_ihashtable.h, 500
hpss_ilist.h, 500
hpss_ilist_foreach_retval, 504
hpss_ilist_foreach_retval_e, 504
hpss_ilist_alloc_func_t
Data Containers, 291
hpss_ilist_append
Data Containers, 293
hpss_ilist_clear
Data Containers, 294
hpss_ilist_custom_init
Data Containers, 294
hpss_ilist_destructor_t
Data Containers, 291
hpss_ilist_empty
Data Containers, 294
hpss_ilist_foreach
Data Containers, 296
hpss_ilist_foreach_range
Data Containers, 296
hpss_ilist_foreach_retval
hpss_ilist.h, 504
hpss_ilist_foreach_retval_e
hpss_ilist.h, 504
hpss_ilist_get_current
Data Containers, 296
hpss_ilist_get_current_r
Data Containers, 297
hpss_ilist_get_node
Data Containers, 297
hpss_ilist_get_size
Data Containers, 297

hpss_ilist_halt_iterator
    Data Containers, 297
hpss_ilist_init
    Data Containers, 298
hpss_ilist_init_iterator
    Data Containers, 298
hpss_ilist_init_iterator_r
    Data Containers, 298
hpss_ilist_init_iterator_range
    Data Containers, 299
hpss_ilist_init_iterator_range_r
    Data Containers, 299
hpss_ilist_insert
    Data Containers, 299
hpss_ilist_iterator_token_h
    Data Containers, 291
hpss_ilist_join
    Data Containers, 300
hpss_ilist_mapped_func_t
    Data Containers, 292
hpss_ilist_prepend
    Data Containers, 300
hpss_ilist_reinit_iterator
    Data Containers, 300
hpss_ilist_reinit_iterator_range
    Data Containers, 301
hpss_ilist_remove
    Data Containers, 301
hpss_ilist_t
    Data Containers, 292
hpss_irbtree.c, 504
hpss_irbtree.h, 506
hpss_irbtree_clear
    Data Containers, 301
hpss_irbtree_color
    Data Containers, 293
hpss_irbtree_container
    Data Containers, 291
hpss_irbtree_empty
    Data Containers, 302
hpss_irbtree_find
    Data Containers, 302
hpss_irbtree_find_cmp
    Data Containers, 302
hpss_irbtree_foreach
    Data Containers, 303
hpss_irbtree_foreach_node
    Data Containers, 303
hpss_irbtree_foreach_node_prev
    Data Containers, 304
hpss_irbtree_init
    Data Containers, 304
hpss_irbtree_insert
    Data Containers, 304
hpss_irbtree_insert_multi
    Data Containers, 305
hpss_irbtree_lower_bound
    Data Containers, 305
hpss_irbtree_max
    Data Containers, 305
hpss_irbtree_min
    Data Containers, 305
hpss_irbtree_node_callback_t
    Data Containers, 292
hpss_irbtree_node_comparator_t
    Data Containers, 292
hpss_irbtree_node_destructor_t
    Data Containers, 292
hpss_irbtree_node_next
    Data Containers, 306
hpss_irbtree_node_prev
    Data Containers, 306
hpss_irbtree_node_printer_t
    Data Containers, 293
hpss_irbtree_print
    Data Containers, 306
hpss_irbtree_remove
    Data Containers, 307
hpss_irbtree_size
    Data Containers, 307
hpss_irbtree_upper_bound
    Data Containers, 307
hpss_migr_order_type_e
    hpss_types.x, 548
hpss_net.c, 509
    __hpss_net_maskaddr32, 510
    __hpss_net_setscopeid, 511
    hpss_net_getaddrinfo, 511
    hpss_net_getaddrinfo_ex, 512
    hpss_net_getsockname, 512
hpss_net.h, 513
    __hpss_net_maskaddr32, 514
    __hpss_net_setscopeid, 514
    hpss_net_getaddrinfo, 515
    hpss_net_getaddrinfo_ex, 515
    hpss_net_getsockname, 516
hpss_net_accept
    HPSS Networking Library, 246
hpss_net_addrmatch
    HPSS Networking Library, 247
hpss_net_bind
    HPSS Networking Library, 247
hpss_net_connect
    HPSS Networking Library, 247
hpss_net_generatemask
    HPSS Networking Library, 249
hpss_net_get_family_option
    HPSS Networking Library, 249
hpss_net_get_family_string
    HPSS Networking Library, 249
hpss_net_getaddrinfo
    hpss_net.c, 511
    hpss_net.h, 515
hpss_net_getaddrinfo_ex
    hpss_net.c, 512
    hpss_net.h, 515

hpss_net_getinaddr
  HPSS Networking Library, 250
hpss_net_getnameinfo
  HPSS Networking Library, 250
hpss_net_getpeername
  HPSS Networking Library, 251
hpss_net_getport
  HPSS Networking Library, 251
hpss_net_getsockname
  hpss_net.c, 512
  hpss_net.h, 516
hpss_net_getuniversaladdress
  HPSS Networking Library, 251
hpss_net_initaddr
  HPSS Networking Library, 252
hpss_net_listen
  HPSS Networking Library, 252
hpss_net_setport
  HPSS Networking Library, 252
hpss_net_socket
  HPSS Networking Library, 253
hpss_net_strerror
  HPSS Networking Library, 253
hpss_net_universaladdresstoport
  HPSS Networking Library, 253
hpss_pio_operation_t
  Client API, 83
hpss_pio_options_t
  Client API, 83
hpss_pio_transport_t
  Client API, 83
hpss_posix_api.h, 516
hpss_readdir_flags_t
  hpss_api.h, 482
hpss_request_state_t
  hpss_types.x, 549
hpss_request_type_t
  hpss_types.x, 549
hpss_rpc_auth_type_t
  hpss_types.x, 549
hpss_rpc_cred_type_t
  hpss_types.x, 549
hpss_rpc_prot_level_t
  hpss_types.x, 550
hpss_sec_api.x, 517
hpss_srvr_id_t
  hpss_types.x, 547
hpss_stage_status_type_t
  core_api_def.x, 460
hpss_stat.h, 518
hpss_string_cmp
  hpss_String.c, 530
hpss_string_fromCstr
  hpss_String.c, 530
hpss_stringbuf_clear
  hpss_String.c, 530
hpss_stringbuf_cmp
  hpss_String.c, 530
hpss_stringbuf_delBackward
  hpss_String.c, 531
hpss_stringbuf_delForward
  hpss_String.c, 531
hpss_stringbuf_free
  hpss_String.c, 531
hpss_stringbuf_fromCstr
  hpss_String.c, 532
hpss_stringbuf_fromFile
  hpss_String.c, 532
hpss_stringbuf_fromString
  hpss_String.c, 532
hpss_stringbuf_grow
  hpss_String.c, 532
hpss_stringbuf_insert
  hpss_String.c, 533
hpss_stringbuf_printf
  hpss_String.c, 533
hpss_tape_schedule_method_t
  hpss_types.x, 550
hpss_time_string
  hpss_String.c, 535
hpss_types.x, 545
  acct_rec_t, 547
  HPSS_OFF_t, 547
  hpss_authn_mech_t, 548
  hpss_authz_mech_t, 548
  hpss_db_partition_t, 547
  hpss_distributionkey_t, 547
  hpss_hash_enum, 548
  hpss_hash_type_last, 550
  hpss_id_t, 547
  hpss_migr_order_type_e, 548
  hpss_request_state_t, 549
  hpss_request_type_t, 549
  hpss_rpc_auth_type_t, 549
  hpss_rpc_cred_type_t, 549
  hpss_rpc_prot_level_t, 550
  hpss_srvr_id_t, 547
  hpss_tape_schedule_method_t, 550
  lbp_verify_error_enum, 550
  size_t, 547
  timespec_t, 547
  timestamp_t, 547
  uchar, 548
  unsigned32, 548
hpss_undelete_flags_t
  Client API, 83
hpss_userattr_t
  Client API, 83
hpss_vattr.h, 550
hpss_xml.c, 551
hpss_xml.h, 552
hpsscfg_config_api.c, 552
hpsscfgx_ConfGetClientInterfaces
  HPSS Client API Extensions, 270
hpsscfgx_ConfParse
  HPSS Client API Extensions, 271

HPSS Config File API, 325
hpsscfgx_ConfSetDirPaths
HPSS Client API Extensions, 271
hpsscfgx_ConfSetFileName
HPSS Client API Extensions, 272
hpsscfgx_DisposeConf
Main, 276
hpsscfgx_FormatMessage
Main, 276
hpsscfgx_GetClientInterfaces.c, 554
hpsscfgx_LookupHostName
HPSS Client API Extensions, 272
hpsscfgx_NetoptFindEntry
HPSS Client API Extensions, 273
hpsscfgx_NetoptGetWriteSize
HPSS Client API Extensions, 273
hpsscfgx_NetoptSetSock
HPSS Client API Extensions, 273
hpsscfgx_ParseTemplate
Main, 276
hpsscfgx_PatternMatch
HPSS Client API Extensions, 274
hpsscfgx_ReadConf
Main, 277
hpsscfgx_ReadConfTemplate
Main, 277
hpsscfgx_cfg_functions.c, 553
hpsscfgx_hpssconf.c, 554
hpsscfgx_pattern_match.c, 555
hpsscfgx_restricted_ports.c, 556
hpssclntaddr.x, 557

IRBTREE_BLACK
Data Containers, 293
IRBTREE_RED
Data Containers, 293
inc64m
64-bit Math Library, 266

LabelTypeStrings
hpss_String.c, 539
lbp_verify_error_block_guard
hpss_types.x, 550
lbp_verify_error_block_method
hpss_types.x, 550
lbp_verify_error_errno
hpss_types.x, 550
lbp_verify_error_none
hpss_types.x, 550
lbp_verify_error_enum
hpss_types.x, 550
le64m
64-bit Math Library, 266
LinkCount
hpss_Attrs, 354
low32m
64-bit Math Library, 266
lt64m
64-bit Math Library, 266

MAX_LOAD_FACTOR
Data Containers, 293
MAX_PATTERN_BUF
hpsscfgx_pattern_match.c, 556
MAX_XPATH_SIZE
Client API, 82
Main, 276
hpsscfgx_DisposeConf, 276
hpsscfgx_FormatMessage, 276
hpsscfgx_ParseTemplate, 276
hpsscfgx_ReadConf, 277
hpsscfgx_ReadConfTemplate, 277
mem64m
64-bit Math Library, 266
mod2x64m
64-bit Math Library, 266
mod64_3m
64-bit Math Library, 266
mod64m
64-bit Math Library, 266
ModePerms
hpss_Attrs, 354
mul64_3m
64-bit Math Library, 266
mul64m
64-bit Math Library, 267

NAMESPEC_GROUP
Client API, 84
NAMESPEC_REALM
Client API, 84
NAMESPEC_SKIP
Client API, 84
NAMESPEC_USER
Client API, 84
NOTRUNC_CLEAR
Client API, 84
NOTRUNC_SET
Client API, 84
namespec_type_t
Client API, 83
neq64m
64-bit Math Library, 267
neqz64m
64-bit Math Library, 267
netaddress, 557
NormalizePath
api_path.c, 426
not64m
64-bit Math Library, 267
notrunc_flag_t
Client API, 84
ns_ACLConfArray, 558
ns_ACLEntry, 559
ns_DirEntryConfArray, 559
ns_FilesetAttrs, 559
ns_FilesetAttrsConfArray, 560
ns_JunctionPathEntryConfArray, 560
ns_ObjHandle, 563

ns_RumbleEntryConfArray, 561
ns_TrashEntryConfArray, 562
ns_TrashcanSettings, 561
ns_FilesetAttrBits_t
    ns_interface_defs.x, 563
ns_ObjHandle.x, 563
ns_interface_defs.x, 557
    ns_FilesetAttrBits_t, 563
NumAdminStateStrings
    hpss_String.c, 539
NumAuthenticatorTypeStrings
    hpss_String.c, 539
NumAuthnMechStrings
    hpss_String.c, 539
NumAuthzMechStrings
    hpss_String.c, 539
NumDriveFlagStrings
    hpss_String.c, 540
NumDriveStateStrings
    hpss_String.c, 540
NumErrnoStrings
    hpss_String.c, 540
NumLabelTypeStrings
    hpss_String.c, 540
NumOpStateStrings
    hpss_String.c, 540
NumPVLActivityStateStrings
    hpss_String.c, 541
NumPVLJobStateStrings
    hpss_String.c, 541
NumPVRTypeStrings
    hpss_String.c, 541
NumProtLevelStrings
    hpss_String.c, 540
NumServerSubtypeStrings
    hpss_String.c, 541
NumServerTypeStrings
    hpss_String.c, 541
NumUsageStateStrings
    hpss_String.c, 541

object_id_hash, 460
object_id_hash_list, 460
object_id_list, 460
OpStateStrings
    hpss_String.c, 542
OpenCount
    hpss_Attrs, 354
OptionFlags
    hpss_Attrs, 354
or64m
    64-bit Math Library, 267
orbit64m
    64-bit Math Library, 267
OtherPerms
    hpss_Attrs, 354

PURGE_LOCK
    Client API, 84
PURGE_ON_MIGRATE_CLEAR
    Client API, 84
PURGE_ON_MIGRATE_SET
    Client API, 84
PURGE_UNLOCK
    Client API, 84
PIOInit
    api_pio.c, 427
POSIX APIs, 282
    hpss_CreateWithHints, 282
    hpss_OpenWithHints, 283
    hpss_PosixCreate, 283
    hpss_PosixFstat, 283
    hpss_PosixLseek, 284
    hpss_PosixLstat, 284
    hpss_PosixOpen, 284
    hpss_PosixStat, 286
PVLActivityStateStrings
    hpss_String.c, 542
PVLJobStateStrings
    hpss_String.c, 542
PVRTypeStrings
    hpss_String.c, 543
Parallel I/O, 232
    hpss_PIOEnd, 232
    hpss_PIOExecute, 233
    hpss_PIOExecuteItem, 233
    hpss_PIOExportGrp, 234
    hpss_PIOFinalize, 234
    hpss_PIOGetPartError, 235
    hpss_PIOGetRequestID, 235
    hpss_PIOImportGrp, 235
    hpss_PIORegister, 236
    hpss_PIOStart, 236
Parsers, 278
    HPSSCFGX_ENUM, 279
    HPSSCFGX_STRING, 279
ProtLevelStrings
    hpss_String.c, 542
purgelock_flag_t
    Client API, 84
purgeonmigrate_flag_t
    Client API, 84
pv_list, 564
pv_list_element, 564

ReadCount
    hpss_Attrs, 354
RealmId
    hpss_Attrs, 354
RegisterBitMap
    hpss_Attrs, 354
RumbleEntryTag, 562

s32_and_map
    hpss_String.c, 536
s32_eq_map
    hpss_String.c, 536
SUPER_PURGE_LOCK

Client API, 84
SUPER_PURGE_UNLOCK
Client API, 84
ServerSubtypeStrings
hpss_String.c, 543
ServerTypeStrings
hpss_String.c, 543
shl64_3m
64-bit Math Library, 267
shl64_ipm
64-bit Math Library, 267
shl64m
64-bit Math Library, 267
shr64_3m
64-bit Math Library, 267
shr64_ipm
64-bit Math Library, 268
shr64m
64-bit Math Library, 268
size_t
hpss_types.x, 547
Special, 280
ss_pvlist.x, 564
stale_thresh_cmp_factor
api_cache.c, 369
sub64_3m
64-bit Math Library, 268
sub64m
64-bit Math Library, 268
SubSystemId
hpss_Attrs, 354
subsys_stats, 78

thresh_cmp
api_cache.c, 369
TimeCreated
hpss_Attrs, 355
TimeLastRead
hpss_Attrs, 355
TimeLastWritten
hpss_Attrs, 355
TimeModified
hpss_Attrs, 355
timespec_t
hpss_types.x, 547
timestamp_t
hpss_types.x, 547
TrashEntryTag, 562
TrashInfo
hpss_Attrs, 355
Trashcan, 238
hpss_GetTrashSettings, 239
hpss_GetTrashcan, 238
hpss_GetTrashcanHandle, 239
hpss_ReadTrash, 240
hpss_Undelete, 240
hpss_UndeleteHandle, 241
Type
hpss_Attrs, 355

u32_and_map
hpss_String.c, 537
u32_eq_map
hpss_String.c, 537
u_signed64.h, 565
CS_DEC64_LEN, 566
UDA_API_VALUE
Client API, 82
UDA_API_XML
Client API, 82
UID
hpss_Attrs, 355
uchar
hpss_types.x, 548
uda_string, 460
unsigned32
hpss_types.x, 548
UsageStateStrings
hpss_String.c, 544
User-defined Attributes, 217
hpss_UserAttrCloseCursor, 218
hpss_UserAttrDeleteAttrHandle, 218
hpss_UserAttrDeleteAttrs, 219
hpss_UserAttrGetAttrHandle, 219
hpss_UserAttrGetAttrs, 220
hpss_UserAttrGetObj, 220
hpss_UserAttrGetObjQuery, 221
hpss_UserAttrGetXML, 221
hpss_UserAttrGetXMLObj, 222
hpss_UserAttrListAttrs, 223
hpss_UserAttrReadObj, 223
hpss_UserAttrReadXML, 224
hpss_UserAttrReadXMLObj, 224
hpss_UserAttrSetAttrHandle, 225
hpss_UserAttrSetAttrs, 225
hpss_UserAttrXQueryGet, 226
hpss_UserAttrXQueryGetHandle, 226
hpss_UserAttrXQueryUpdate, 227
hpss_UserAttrXQueryUpdateHandle, 227
UserPerms
hpss_Attrs, 355
userkeycompare
api_cache.c, 369

Validators, 281

Working Directory, 189
hpss_Chdir, 189
hpss_Chroot, 189
hpss_GetThreadUcred, 190
hpss_Getcwd, 190
WriteCount
hpss_Attrs, 355

XML_ATTR
Client API, 82
XML_NO_ATTR
Client API, 82