

HPSS 11.3 Programmer's Reference

Generated on Tue Sep 23 2025

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 PIO Simplified APIs	8
3.1.9 Trashcans	8
3.1.9.1 Getting Information on Trashcans	8
3.1.9.2 Limitations, Caveats, and Warnings	8
3.2 64-bit Arithmetic Library	9
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
4.2.2.1 Client API Binary Package	15
4.2.2.2 Client API Development Package	15
4.2.2.3 Client API Mover Protocol Development Package	15
4.2.3 Install from Source	15
4.2.3.1 Extraction from HPSS Source Tree	15
4.2.3.2 Compilation	16
4.2.4 Using pkg-config To Compile	16
4.2.5 Testing	16
4.2.6 Configuration	17
4.2.7 Client API Environment Variables	17
4.2.8 Client API Runtime Configuration	19
4.2.9 Other System Configuration	20
4.3 Tuning and Troubleshooting	20
4.3.1 Tuning	20
4.3.2 Expectations	20
4.3.2.1 Tape	20
4.3.2.2 Metadata	20
4.3.3 Testing Procedures	20
4.3.4 Tuning Concepts	21
4.3.5 Troubleshooting	21
4.3.5.1 Debug and Trace Levels	21
4.3.5.2 API Logs	22
4.3.5.3 HPSS Logs	23
4.3.5.4 Crash Dumps	23
4.4 Client API Uses	23
4.4.1 Overview and Applications	23
4.5 Tutorials	23
4.5.1 Overview	24
4.5.2 A Short Example: Current Working Directory	24
4.5.3 Namespace Metadata	26
4.5.3.1 Create	27
4.5.3.2 Rename	27
4.5.3.3 Unlink	27
4.5.3.4 Get Attributes	28
4.5.3.5 Get Extended Attributes	28
4.5.3.6 Directory Listing	29
4.5.3.7 Short Listing	29
4.5.3.8 Long Listing	30
4.5.4 File Input and Output	31
4.5.4.1 hpss_Write	31
4.5.4.2 hpss_Read	31

4.5.4.3 PIO Concepts	32
4.5.5 HSM Functions	35
4.5.5.1 Migrate	35
4.5.5.2 Purge	36
4.5.5.3 Purge Lock	36
4.5.5.4 Stage	37
4.5.6 User-defined Attributes (UDA)	37
4.5.6.1 Set UDAs	38
4.5.6.2 Get UDA	38
4.5.6.3 List UDA	39
4.5.6.4 Delete UDA	39
4.5.6.5 Using Custom XQueries	40
4.5.6.6 Using Custom XQueries for Retrieval	41
4.5.6.7 Searching UDA	41
4.5.7 Using UDAs with VFS	42
4.5.8 HPSS to POSIX and Back Again	43
4.5.8.1 HPSS Mode to POSIX Mode	43
4.5.8.2 HPSS Time to POSIX Time	43
4.5.9 More On UDAs	44
4.5.9.1 Guidelines, Best Practices, and Standard Attributes	44
4.5.10 Useful References	45
5 Developer Acknowledgements	47
6 Further Reading	49
7 Glossary of Terms and Acronyms	51
8 Deprecated List	59
9 Module Documentation	61
9.1 Client API	61
9.1.1 Detailed Description	78
9.1.2 Class Documentation	78
9.1.2.1 struct api_config	78
9.1.2.2 struct api_dist_file_info_t	80
9.1.2.3 struct api_namespec	80
9.1.2.4 struct hpss_errno_state	80
9.1.2.5 struct hpss_file_hash_digest	81
9.1.2.6 struct hpss_file_hash_digest_list	81
9.1.2.7 struct hpss_global_fsent_t	81
9.1.2.8 struct hpss_junction_ent_t	81
9.1.2.9 struct hpss_pio_gapinfo_s	82
9.1.2.10 struct hpss_pio_params_s	82

9.1.2.11 struct hpss_stat	82
9.1.2.12 struct hpss_statfs	83
9.1.2.13 struct hpss_statfs64	83
9.1.2.14 struct hpss_statvfs	84
9.1.2.15 struct hpss_statvfs64	84
9.1.2.16 struct hpss_userattr	84
9.1.2.17 struct hpss_userattr_list	85
9.1.2.18 struct subsys_stats	85
9.1.3 Macro Definition Documentation	85
9.1.3.1 API_CONFIG_ALL_FLAGS	85
9.1.3.2 API_CONFIG_DEFAULT	85
9.1.3.3 API_DEBUG_ERROR	86
9.1.3.4 API_DEBUG_NONE	86
9.1.3.5 API_DEBUG_REQUEST	86
9.1.3.6 API_DEBUG_TRACE	86
9.1.3.7 API_DISABLE_CROSS_REALM	86
9.1.3.8 API_DISABLE_JUNCTIONS	86
9.1.3.9 API_ENABLE_LOGGING	86
9.1.3.10 API_MSGTYPE_MVRPROT	86
9.1.3.11 API_MSGTYPE_PDATA	87
9.1.3.12 API_TRANSFER_MVRSELECT	87
9.1.3.13 API_TRANSFER_TCP	87
9.1.3.14 API_USE_CONFIG	87
9.1.3.15 API_USE_SAN3P	87
9.1.3.16 HPSS_MIGRATE_FORCE	87
9.1.3.17 HPSS_MIGRATE_HPSS_ONLY	87
9.1.3.18 HPSS_MIGRATE_NO_COPY	87
9.1.3.19 HPSS_MIGRATE_PURGE_DATA	88
9.1.3.20 HPSS_O_EXCL	88
9.1.3.21 HPSS_O_NO_TAPE	88
9.1.3.22 HPSS_O_STAGE_ASYNC	88
9.1.3.23 HPSS_O_STAGE_BKGRD	88
9.1.3.24 HPSS_O_STAGE_NONE	88
9.1.3.25 HPSS_PIO_HANDLE_GAP	88
9.1.3.26 HPSS_PIO_PORT_RANGE	89
9.1.3.27 HPSS_PIO_PUSH	89
9.1.3.28 HPSS_PIO_USE_API_HOSTNAME	89
9.1.3.29 HPSS_STAGE_STATUS_ACTIVE	89
9.1.3.30 HPSS_STAGE_STATUS_QUEUED	89
9.1.3.31 HPSS_STAGE_STATUS_UNKNOWN	89
9.1.3.32 MAX_XPATH_ELEM_SIZE	89
9.1.3.33 MAX_XPATH_SIZE	89

9.1.3.34 UDA_API_VALUE	90
9.1.3.35 UDA_API_XML	90
9.1.3.36 XML_ATTR	90
9.1.3.37 XML_NO_ATTR	90
9.1.4 Typedef Documentation	90
9.1.4.1 api_namespec_t	90
9.1.4.2 hpss_errno_state_t	90
9.1.4.3 HPSS_FILE	91
9.1.4.4 hpss_file_hash_flags_t	91
9.1.4.5 hpss_pio_options_t	91
9.1.4.6 hpss_userattr_t	91
9.1.5 Enumeration Type Documentation	91
9.1.5.1 hpss_file_hash_flags	91
9.1.5.2 hpss_file_hash_stripe_flags	92
9.1.5.3 hpss_pio_operation_t	92
9.1.5.4 hpss_pio_transport_t	92
9.1.5.5 hpss_readdir_flags_t	93
9.1.5.6 hpss_undelete_flags_t	93
9.1.5.7 namespec_type_t	93
9.1.5.8 notrunc_flag_t	94
9.1.5.9 purgelock_flag_t	94
9.1.5.10 purgeonmigrate_flag_t	94
9.1.6 Function Documentation	95
9.1.6.1 API_AddAllRegisterValues()	95
9.1.6.2 API_AddRegisterValues()	95
9.1.6.3 API_CalcBlock32()	96
9.1.6.4 API_CalcBlock64()	96
9.1.6.5 API_ConvertAttrsToVAttrs()	97
9.1.6.6 API_ConvertModeToPosixMode()	97
9.1.6.7 API_ConvertPosixModeToMode()	98
9.1.6.8 API_ConvertPosixTimeToTime()	98
9.1.6.9 API_ConvertTimeToPosixTime()	99
9.1.6.10 API_FreeIOR()	99
9.1.6.11 API_GetBatchStatus()	100
9.1.6.12 API_GetClientAddr()	100
9.1.6.13 API_GetDataThreadStackSize()	101
9.1.6.14 API_GetServerIDForSubsystem()	101
9.1.6.15 API_GetUniqueRequestID()	102
9.1.6.16 API_RemoveRegisterValues()	102
9.1.6.17 API_StageBatchFree()	102
9.1.6.18 API_StageBatchGetEntry()	103
9.1.6.19 API_StageBatchInit()	103

9.1.6.20 API_StageBatchInsertBFHandle()	104
9.1.6.21 API_StageBatchInsertBFObj()	105
9.1.6.22 API_StageBatchInsertFd()	106
9.1.6.23 API_StageStatusFree()	107
9.1.6.24 hpss_AbortIOByMoverId()	107
9.1.6.25 hpss_AbortIOByMoverName()	107
9.1.6.26 hpss_AbortIOByRequestId()	108
9.1.6.27 hpss_AbortIOByRequestMatch()	108
9.1.6.28 hpss_AbortIOByUserId()	110
9.1.6.29 hpss_Access()	111
9.1.6.30 hpss_AccessHandle()	112
9.1.6.31 hpss_AcctCodeToName()	112
9.1.6.32 hpss_AcctNameToCode()	113
9.1.6.33 hpss_BuildLevelString()	114
9.1.6.34 hpss_Chacct()	114
9.1.6.35 hpss_ChacctByName()	115
9.1.6.36 hpss_Chdir()	115
9.1.6.37 hpss_Chmod()	117
9.1.6.38 hpss_Chown()	117
9.1.6.39 hpss_Chroot()	118
9.1.6.40 hpss_ClearLastHPSSErrno()	119
9.1.6.41 hpss_ClientAPIInit()	119
9.1.6.42 hpss_ClientAPIReset()	119
9.1.6.43 hpss_Close()	120
9.1.6.44 hpss_Closedir()	120
9.1.6.45 hpss_ConvertIdsToNames()	121
9.1.6.46 hpss_ConvertNamesToIds()	122
9.1.6.47 hpss_CopyFile()	123
9.1.6.48 hpss_CopyrightStrings()	123
9.1.6.49 hpss_Creat()	124
9.1.6.50 hpss_Create()	124
9.1.6.51 hpss_CreateHandle()	125
9.1.6.52 hpss_DeleteACL()	126
9.1.6.53 hpss_DeleteACLHandle()	127
9.1.6.54 hpss_Fclear()	128
9.1.6.55 hpss_FclearOffset()	128
9.1.6.56 hpss_Fclose()	130
9.1.6.57 hpss_Fcntl()	131
9.1.6.58 hpss_FdelFileDigestList()	132
9.1.6.59 hpss_Fflush()	133
9.1.6.60 hpss_Fgetc()	133
9.1.6.61 hpss_FgetFileDigest()	134

9.1.6.62 hpss_FgetFileDigestList()	135
9.1.6.63 hpss_Fgets()	136
9.1.6.64 hpss_FileGetAttributes()	136
9.1.6.65 hpss_FileGetAttributesBitfile()	137
9.1.6.66 hpss_FileGetAttributesHandle()	138
9.1.6.67 hpss_FileGetDigestHandle()	139
9.1.6.68 hpss_FileGetDigestListHandle()	140
9.1.6.69 hpss_FileGetXAttributes()	141
9.1.6.70 hpss_FileGetXAttributesHandle()	142
9.1.6.71 hpss_FileSetAttributes()	143
9.1.6.72 hpss_FileSetAttributesBitfile()	145
9.1.6.73 hpss_FileSetAttributesHandle()	146
9.1.6.74 hpss_FileSetCOS()	147
9.1.6.75 hpss_FileSetCOSHandle()	148
9.1.6.76 hpss_FilesetCreate()	149
9.1.6.77 hpss_FilesetDelete()	150
9.1.6.78 hpss_FilesetGetAttributes()	150
9.1.6.79 hpss_FilesetListAll()	151
9.1.6.80 hpss_FilesetSetAttributes()	152
9.1.6.81 hpss_Fopen()	153
9.1.6.82 hpss_Fpreallocate()	154
9.1.6.83 hpss_Fread()	155
9.1.6.84 hpss_Fseek()	156
9.1.6.85 hpss_FsetFileDigest()	156
9.1.6.86 hpss_FsetFileDigestList()	157
9.1.6.87 hpss_Fstat()	158
9.1.6.88 hpss_Fsync()	159
9.1.6.89 hpss_Ftell()	159
9.1.6.90 hpss_Ftruncate()	160
9.1.6.91 hpss_Fwrite()	160
9.1.6.92 hpss_GetAcct()	161
9.1.6.93 hpss_GetAcctName()	162
9.1.6.94 hpss_GetACL()	162
9.1.6.95 hpss_GetACLHandle()	163
9.1.6.96 hpss_GetAllSubsystems()	164
9.1.6.97 hpss_GetAsyncStatus()	164
9.1.6.98 hpss_GetAttrHandle()	165
9.1.6.99 hpss_GetBatchAsynchStatus()	166
9.1.6.100 hpss_GetCallbackStatus()	166
9.1.6.101 hpss_GetConfiguration()	167
9.1.6.102 hpss_Getcwd()	167
9.1.6.103 hpss_Getdents()	168

9.1.6.104 hpss_GetdentsAttrs()	169
9.1.6.105 hpss_GetDistFile()	169
9.1.6.106 hpss_GetFileNotrunc()	170
9.1.6.107 hpss_GetFullPath()	170
9.1.6.108 hpss_GetFullPathBitfile()	171
9.1.6.109 hpss_GetFullPathBitfileLen()	172
9.1.6.110 hpss_GetFullPathHandle()	172
9.1.6.111 hpss_GetFullPathHandleLen()	173
9.1.6.112 hpss_GetFullPathLen()	173
9.1.6.113 hpss_GetGID()	174
9.1.6.114 hpss_GetIntersectedList()	174
9.1.6.115 hpss_GetJunctionAttrs()	175
9.1.6.116 hpss_GetJunctions()	175
9.1.6.117 hpss_GetLastHPSS_errno()	176
9.1.6.118 hpss_GetListAttrs()	176
9.1.6.119 hpss_GetNextIORquestID()	177
9.1.6.120 hpss_GetObjFromHandle()	177
9.1.6.121 hpss_GetObjType()	178
9.1.6.122 hpss_GetPathHandle()	178
9.1.6.123 hpss_GetPathHandleObjHash()	179
9.1.6.124 hpss_GetPathObjHash()	179
9.1.6.125 hpss_GetRawAttrHandle()	180
9.1.6.126 hpss_GetSubSysLimits()	181
9.1.6.127 hpss_GetSubSysStats()	181
9.1.6.128 hpss_GetThreadUcred()	182
9.1.6.129 hpss_GetTrashcan()	182
9.1.6.130 hpss_GetTrashcanHandle()	183
9.1.6.131 hpss_GetTrashSettings()	184
9.1.6.132 hpss_GetUID()	184
9.1.6.133 hpss_HashFlagsToString()	185
9.1.6.134 hpss_InsertDistFile()	185
9.1.6.135 hpss_JunctionCreate()	186
9.1.6.136 hpss_JunctionCreateHandle()	186
9.1.6.137 hpss_JunctionDelete()	187
9.1.6.138 hpss_JunctionDeleteHandle()	187
9.1.6.139 hpss_Link()	188
9.1.6.140 hpss_LinkHandle()	189
9.1.6.141 hpss_LinkHandleParent()	190
9.1.6.142 hpss_LoadDefaultThreadState()	190
9.1.6.143 hpss_LoadThreadState()	191
9.1.6.144 hpss_LocateRootCS()	192
9.1.6.145 hpss_LocateServerByID()	192

9.1.6.146 hpss_LocateSiteById()	193
9.1.6.147 hpss_LocateSiteByName()	193
9.1.6.148 hpss_LookupRootCS()	194
9.1.6.149 hpss_Lseek()	194
9.1.6.150 hpss_Lstat()	195
9.1.6.151 hpss_Migrate()	196
9.1.6.152 hpss_MiscAdmin()	196
9.1.6.153 hpss_Mkdir()	198
9.1.6.154 hpss_MkdirHandle()	198
9.1.6.155 hpss_Open()	199
9.1.6.156 hpss_OpenBitfile()	200
9.1.6.157 hpss_OpenBitfileVAttr()	202
9.1.6.158 hpss_Opendir()	204
9.1.6.159 hpss_OpendirHandle()	204
9.1.6.160 hpss_OpenHandle()	205
9.1.6.161 hpss_PingCore()	206
9.1.6.162 hpss_PIOEnd()	207
9.1.6.163 hpss_PIOExecute()	208
9.1.6.164 hpss_PIOExecuteItem()	208
9.1.6.165 hpss_PIOExportGrp()	209
9.1.6.166 hpss_PIOFinalize()	210
9.1.6.167 hpss_PIOGetPartError()	210
9.1.6.168 hpss_PIOGetRequestID()	211
9.1.6.169 hpss_PIOImportGrp()	211
9.1.6.170 hpss_PIOInitCtx()	212
9.1.6.171 hpss_PIOReadFile()	212
9.1.6.172 hpss_PIOReadFileMulti()	213
9.1.6.173 hpss_PIORegister()	214
9.1.6.174 hpss_PIOResetCtx()	214
9.1.6.175 hpss_PIOSetBlockSize()	215
9.1.6.176 hpss_PIOSetDefaultCtx()	215
9.1.6.177 hpss_PIOSetHandleGaps()	216
9.1.6.178 hpss_PIOSetStripeWidth()	216
9.1.6.179 hpss_PIOSetUseAPIHost()	217
9.1.6.180 hpss_PIOStart()	217
9.1.6.181 hpss_PIOWriteFile()	218
9.1.6.182 hpss_PIOWriteFileMulti()	218
9.1.6.183 hpss_PreferedReadOrder()	219
9.1.6.184 hpss_Purge()	220
9.1.6.185 hpss_PurgeLock()	221
9.1.6.186 hpss_PurgeLoginCred()	221
9.1.6.187 hpss_PurgeOnMigrate()	222

9.1.6.188 hpss_Read()	222
9.1.6.189 hpss_ReadAttrs()	223
9.1.6.190 hpss_ReadAttrsHandle()	224
9.1.6.191 hpss_ReadAttrsPlus()	224
9.1.6.192 hpss_Readdir()	225
9.1.6.193 hpss_ReaddirHandle()	226
9.1.6.194 hpss_ReaddirPlus()	227
9.1.6.195 hpss_ReadItem()	228
9.1.6.196 hpss_Readlink()	228
9.1.6.197 hpss_ReadlinkHandle()	229
9.1.6.198 hpss_ReadList()	230
9.1.6.199 hpss_ReadQueueAdd()	230
9.1.6.200 hpss_ReadQueueAddMany()	231
9.1.6.201 hpss_ReadQueueCreateContext()	231
9.1.6.202 hpss_ReadQueueList()	232
9.1.6.203 hpss_ReadQueueRemove()	232
9.1.6.204 hpss_ReadQueueRemoveContext()	233
9.1.6.205 hpss_ReadRawAttrsHandle()	234
9.1.6.206 hpss_RemoveIntersectionFromList()	234
9.1.6.207 hpss_Rename()	235
9.1.6.208 hpss_RenameHandle()	236
9.1.6.209 hpss_ReopenBitfile()	237
9.1.6.210 hpss_ResetSubSysStats()	238
9.1.6.211 hpss_Rewinddir()	238
9.1.6.212 hpss_Rmdir()	239
9.1.6.213 hpss_RmdirHandle()	239
9.1.6.214 hpss_RmdirHandleImmediate()	240
9.1.6.215 hpss_RmdirImmediate()	241
9.1.6.216 hpss_SetAcct()	242
9.1.6.217 hpss_SetAcctByName()	242
9.1.6.218 hpss_SetACL()	243
9.1.6.219 hpss_SetACLHandle()	244
9.1.6.220 hpss_SetAPILogLevel()	244
9.1.6.221 hpss_SetAPILogPath()	245
9.1.6.222 hpss_SetApplicationName()	245
9.1.6.223 hpss_SetAppLoginCred()	246
9.1.6.224 hpss_SetAttrHandle()	246
9.1.6.225 hpss_SetAuditInfo()	247
9.1.6.226 hpss_SetConfiguration()	248
9.1.6.227 hpss_SetCOSByHints()	248
9.1.6.228 hpss_SetFileNotrunc()	249
9.1.6.229 hpss_SetFileOffset()	250

9.1.6.230 hpss_SetGID()	251
9.1.6.231 hpss_SetLoginCred()	251
9.1.6.232 hpss_SetNextIORRequestID()	252
9.1.6.233 hpss_SiteIdToName()	252
9.1.6.234 hpss_SiteNameToId()	253
9.1.6.235 hpss_Stage()	253
9.1.6.236 hpss_StageBatch()	254
9.1.6.237 hpss_StageBatchCallBack()	255
9.1.6.238 hpss_StageCallBack()	255
9.1.6.239 hpss_StageCallBackBitfile()	257
9.1.6.240 hpss_Stat()	258
9.1.6.241 hpss_Statfs()	258
9.1.6.242 hpss_Statfs64()	259
9.1.6.243 hpss_Statvfs()	259
9.1.6.244 hpss_Statvfs64()	260
9.1.6.245 hpss_Symlink()	261
9.1.6.246 hpss_SymlinkHandle()	261
9.1.6.247 hpss_ThreadCleanUp()	262
9.1.6.248 hpss_Truncate()	262
9.1.6.249 hpss_TruncateHandle()	263
9.1.6.250 hpss_Umask()	264
9.1.6.251 hpss_Undelete()	264
9.1.6.252 hpss_UndeleteHandle()	265
9.1.6.253 hpss_Unlink()	266
9.1.6.254 hpss_UnlinkHandle()	266
9.1.6.255 hpss_UnlinkHandleImmediate()	267
9.1.6.256 hpss_UnlinkImmediate()	268
9.1.6.257 hpss_UpdateACL()	268
9.1.6.258 hpss_UpdateACLHandle()	269
9.1.6.259 hpss_UserAttrCloseCursor()	270
9.1.6.260 hpss_UserAttrDeleteAttrHandle()	270
9.1.6.261 hpss_UserAttrDeleteAttrs()	271
9.1.6.262 hpss_UserAttrGetAttrHandle()	272
9.1.6.263 hpss_UserAttrGetAttrs()	273
9.1.6.264 hpss_UserAttrListAttrHandle()	273
9.1.6.265 hpss_UserAttrListAttrs()	274
9.1.6.266 hpss_UserAttrReadObj()	275
9.1.6.267 hpss_UserAttrReadXML()	276
9.1.6.268 hpss_UserAttrReadXMLObj()	276
9.1.6.269 hpss_UserAttrSetAttrHandle()	277
9.1.6.270 hpss_UserAttrSetAttrs()	278
9.1.6.271 hpss_UserAttrXQueryGet()	279

9.1.6.272 hpss_UserAttrXQueryGetHandle()	279
9.1.6.273 hpss_UserAttrXQueryUpdate()	280
9.1.6.274 hpss_UserAttrXQueryUpdateHandle()	280
9.1.6.275 hpss_Utime()	281
9.1.6.276 hpss_Write()	282
9.1.6.277 hpss_WriteList()	282
9.2 File Open, Create, and Close	283
9.2.1 Detailed Description	284
9.2.2 Function Documentation	284
9.2.2.1 Common_hpss_UnlinkHandle()	284
9.2.2.2 hpss_Close()	285
9.2.2.3 hpss_Creat()	286
9.2.2.4 hpss_Create()	286
9.2.2.5 hpss_CreateHandle()	287
9.2.2.6 hpss_Open()	288
9.2.2.7 hpss_OpenBitfile()	289
9.2.2.8 hpss_OpenBitfileVAttrs()	290
9.2.2.9 hpss_OpenHandle()	292
9.2.2.10 hpss_ReopenBitfile()	293
9.2.2.11 hpss_Unlink()	294
9.2.2.12 hpss_UnlinkHandle()	296
9.2.2.13 hpss_UnlinkHandleImmediate()	297
9.2.2.14 hpss_UnlinkImmediate()	298
9.3 File Data	298
9.3.1 Detailed Description	299
9.3.2 Function Documentation	300
9.3.2.1 hpss_CopyFile()	300
9.3.2.2 hpss_Fclear()	300
9.3.2.3 hpss_FclearOffset()	301
9.3.2.4 hpss_Fpreallocate()	301
9.3.2.5 hpss_Ftruncate()	302
9.3.2.6 hpss_Lseek()	303
9.3.2.7 hpss_PreferedReadOrder()	303
9.3.2.8 hpss_Read()	305
9.3.2.9 hpss_ReadItem()	306
9.3.2.10 hpss_ReadQueueAdd()	307
9.3.2.11 hpss_ReadQueueAddMany()	307
9.3.2.12 hpss_ReadQueueCreateContext()	308
9.3.2.13 hpss_ReadQueueList()	308
9.3.2.14 hpss_ReadQueueRemove()	309
9.3.2.15 hpss_ReadQueueRemoveContext()	309
9.3.2.16 hpss_SetFileOffset()	310

9.3.2.17 hpss_SetNextIORequestID()	310
9.3.2.18 hpss_Truncate()	311
9.3.2.19 hpss_TruncateHandle()	312
9.3.2.20 hpss_Write()	312
9.4 Fileset and Junction	313
9.4.1 Detailed Description	314
9.4.2 Function Documentation	314
9.4.2.1 hpss_FilesetCreate()	314
9.4.2.2 hpss_FilesetDelete()	315
9.4.2.3 hpss_FilesetGetAttributes()	316
9.4.2.4 hpss_FilesetListAll()	317
9.4.2.5 hpss_FilesetSetAttributes()	317
9.4.2.6 hpss_GetJunctionAttrs()	318
9.4.2.7 hpss_GetJunctions()	319
9.4.2.8 hpss_JunctionCreate()	319
9.4.2.9 hpss_JunctionCreateHandle()	320
9.4.2.10 hpss_JunctionDelete()	321
9.4.2.11 hpss_JunctionDeleteHandle()	321
9.5 File Attribute	322
9.5.1 Detailed Description	323
9.5.2 Function Documentation	324
9.5.2.1 hpss_Access()	324
9.5.2.2 hpss_AccessHandle()	324
9.5.2.3 hpss_Chacct()	325
9.5.2.4 hpss_ChacctByName()	326
9.5.2.5 hpss_Chmod()	326
9.5.2.6 hpss_Chown()	327
9.5.2.7 hpss_FileGetAttributes()	328
9.5.2.8 hpss_FileGetAttributesBitfile()	328
9.5.2.9 hpss_FileGetAttributesHandle()	329
9.5.2.10 hpss_FileGetXAttributes()	330
9.5.2.11 hpss_FileGetXAttributesHandle()	331
9.5.2.12 hpss_FileSetAttributes()	332
9.5.2.13 hpss_FileSetAttributesBitfile()	334
9.5.2.14 hpss_FileSetAttributesHandle()	335
9.5.2.15 hpss_FileSetCOS()	336
9.5.2.16 hpss_FileSetCOSHandle()	337
9.5.2.17 hpss_Fstat()	338
9.5.2.18 hpss_GetAttrHandle()	338
9.5.2.19 hpss_GetFileNotrunc()	339
9.5.2.20 hpss_GetListAttrs()	339
9.5.2.21 hpss_GetRawAttrHandle()	340

9.5.2.22 hpss_Lstat()	341
9.5.2.23 hpss_PurgeLock()	341
9.5.2.24 hpss_PurgeOnMigrate()	342
9.5.2.25 hpss_SetAttrHandle()	343
9.5.2.26 hpss_SetCOSByHints()	344
9.5.2.27 hpss_SetFileNotrunc()	344
9.5.2.28 hpss_Stat()	345
9.5.2.29 hpss_Utime()	346
9.6 File Name	346
9.6.1 Detailed Description	347
9.6.2 Function Documentation	347
9.6.2.1 hpss_Link()	347
9.6.2.2 hpss_LinkHandle()	347
9.6.2.3 hpss_LinkHandleParent()	349
9.6.2.4 hpss_Rename()	350
9.6.2.5 hpss_RenameHandle()	351
9.6.2.6 hpss_Symlink()	352
9.6.2.7 hpss_SymlinkHandle()	352
9.7 ACLs	353
9.7.1 Detailed Description	354
9.7.2 Function Documentation	354
9.7.2.1 hpss_DeleteACL()	354
9.7.2.2 hpss_DeleteACLHandle()	355
9.7.2.3 hpss_GetACL()	356
9.7.2.4 hpss_GetACLHandle()	356
9.7.2.5 hpss_SetACL()	357
9.7.2.6 hpss_SetACLHandle()	358
9.7.2.7 hpss_UpdateACL()	359
9.7.2.8 hpss_UpdateACLHandle()	360
9.8 Directory Creation and Deletion	360
9.8.1 Detailed Description	361
9.8.2 Function Documentation	361
9.8.2.1 hpss_Mkdir()	361
9.8.2.2 hpss_MkdirHandle()	362
9.8.2.3 hpss_Rmdir()	363
9.8.2.4 hpss_RmdirHandle()	363
9.8.2.5 hpss_RmdirHandleImmediate()	364
9.8.2.6 hpss_RmdirImmediate()	365
9.9 Directory Data	365
9.9.1 Detailed Description	366
9.9.2 Function Documentation	366
9.9.2.1 hpss_Closedir()	366

9.9.2.2 hpss_Getdents()	367
9.9.2.3 hpss_GetdentsAttrs()	368
9.9.2.4 hpss_Opendir()	369
9.9.2.5 hpss_OpendirHandle()	369
9.9.2.6 hpss_ReadAttrs()	370
9.9.2.7 hpss_ReadAttrsHandle()	370
9.9.2.8 hpss_ReadAttrsPlus()	371
9.9.2.9 hpss_Readdir()	372
9.9.2.10 hpss_ReaddirHandle()	373
9.9.2.11 hpss_ReaddirPlus()	374
9.9.2.12 hpss_ReadRawAttrsHandle()	375
9.9.2.13 hpss_Rewinddir()	376
9.10 Working Directory	376
9.10.1 Detailed Description	376
9.10.2 Function Documentation	376
9.10.2.1 hpss_Chdir()	377
9.10.2.2 hpss_Chroot()	377
9.10.2.3 hpss_Getcwd()	378
9.10.2.4 hpss_GetThreadUcred()	378
9.11 Client API Control	379
9.11.1 Detailed Description	379
9.11.2 Function Documentation	379
9.11.2.1 API_GetDataThreadStackSize()	380
9.11.2.2 API_SocketSetOptions()	380
9.11.2.3 hpss_ClientAPIInit()	380
9.11.2.4 hpss_ClientAPIReset()	382
9.11.2.5 hpss_GetConfiguration()	382
9.11.2.6 hpss_LoadDefaultThreadState()	382
9.11.2.7 hpss_MiscAdmin()	383
9.11.2.8 hpss_SetAPILogLevel()	384
9.11.2.9 hpss_SetAPILogPath()	385
9.11.2.10 hpss_SetApplicationName()	385
9.11.2.11 hpss_SetConfiguration()	386
9.11.2.12 hpss_ThreadCleanup()	386
9.12 Authentication	386
9.12.1 Detailed Description	387
9.12.2 Function Documentation	387
9.12.2.1 hpss_PurgeLoginCred()	387
9.12.2.2 hpss_SetAppLoginCred()	387
9.12.2.3 hpss_SetAuditInfo()	388
9.12.2.4 hpss_SetLoginCred()	389
9.13 HPSS Statistics	389

9.13.1 Detailed Description	390
9.13.2 Function Documentation	390
9.13.2.1 hpss_GetSubSysLimits()	390
9.13.2.2 hpss_GetSubSysStats()	391
9.13.2.3 hpss_ResetSubSysStats()	391
9.13.2.4 hpss_Statfs()	392
9.13.2.5 hpss_Statfs64()	392
9.13.2.6 hpss_Statvfs()	393
9.13.2.7 hpss_Statvfs64()	393
9.14 HPSS Object	394
9.14.1 Detailed Description	394
9.14.2 Function Documentation	394
9.14.2.1 hpss_GetObjFromHandle()	394
9.14.2.2 hpss_GetObjType()	395
9.14.2.3 hpss_GetPathHandle()	395
9.14.2.4 hpss_GetPathHandleObjHash()	396
9.14.2.5 hpss_GetPathObjHash()	396
9.15 Buffered Data	397
9.15.1 Detailed Description	397
9.15.2 Function Documentation	398
9.15.2.1 hpss_Fclose()	398
9.15.2.2 hpss_Fcntl()	398
9.15.2.3 hpss_Fflush()	400
9.15.2.4 hpss_Fgetc()	400
9.15.2.5 hpss_Fgets()	401
9.15.2.6 hpss_Fopen()	402
9.15.2.7 hpss_Fread()	403
9.15.2.8 hpss_Fseek()	404
9.15.2.9 hpss_Fsync()	405
9.15.2.10 hpss_Ftell()	405
9.15.2.11 hpss_Fwrite()	406
9.16 User-defined Attributes	407
9.16.1 Detailed Description	408
9.16.2 Function Documentation	408
9.16.2.1 hpss_UserAttrCloseCursor()	408
9.16.2.2 hpss_UserAttrDeleteAttrHandle()	409
9.16.2.3 hpss_UserAttrDeleteAttrs()	409
9.16.2.4 hpss_UserAttrGetAttrHandle()	410
9.16.2.5 hpss_UserAttrGetAttrs()	411
9.16.2.6 hpss_UserAttrGetObj()	412
9.16.2.7 hpss_UserAttrGetObjQuery()	412
9.16.2.8 hpss_UserAttrGetXML()	413

9.16.2.9 hpss_UserAttrGetXMLObj()	414
9.16.2.10 hpss_UserAttrListAttrs()	415
9.16.2.11 hpss_UserAttrReadObj()	416
9.16.2.12 hpss_UserAttrReadXML()	416
9.16.2.13 hpss_UserAttrReadXMLObj()	417
9.16.2.14 hpss_UserAttrSetAttrHandle()	417
9.16.2.15 hpss_UserAttrSetAttrs()	418
9.16.2.16 hpss_UserAttrXQueryGet()	419
9.16.2.17 hpss_UserAttrXQueryGetHandle()	420
9.16.2.18 hpss_UserAttrXQueryUpdate()	420
9.16.2.19 hpss_UserAttrXQueryUpdateHandle()	421
9.17 Data Structure Conversion	422
9.17.1 Detailed Description	422
9.17.2 Function Documentation	422
9.17.2.1 API_CalcBlock32()	422
9.17.2.2 API_CalcBlock64()	423
9.17.2.3 API_ConvertModeToPosixMode()	423
9.17.2.4 API_ConvertPosixModeToMode()	424
9.17.2.5 API_ConvertPosixTimeToTime()	424
9.17.2.6 API_ConvertTimeToPosixTime()	425
9.18 Parallel I/O	425
9.18.1 Detailed Description	426
9.18.2 Function Documentation	426
9.18.2.1 hpss_PIOEnd()	426
9.18.2.2 hpss_PIOExecute()	427
9.18.2.3 hpss_PIOExecuteItem()	427
9.18.2.4 hpss_PIOExportGrp()	428
9.18.2.5 hpss_PIOFinalize()	429
9.18.2.6 hpss_PIOGetPartError()	429
9.18.2.7 hpss_PIOGetRequestID()	430
9.18.2.8 hpss_PIOImportGrp()	430
9.18.2.9 hpss_PIORegister()	431
9.18.2.10 hpss_PIOStart()	432
9.19 Simplified Parallel I/O	432
9.19.1 Detailed Description	433
9.19.2 Function Documentation	433
9.19.2.1 hpss_PIOInitCtx()	433
9.19.2.2 hpss_PIOReadFile()	434
9.19.2.3 hpss_PIOReadFileMulti()	434
9.19.2.4 hpss_PIOResetCtx()	435
9.19.2.5 hpss_PIOSetBlockSize()	435
9.19.2.6 hpss_PIOSetDefaultCtx()	436

9.19.2.7 <code>hpss_PIOSetHandleGaps()</code>	437
9.19.2.8 <code>hpss_PIOSetOffset()</code>	437
9.19.2.9 <code>hpss_PIOSetStripeWidth()</code>	437
9.19.2.10 <code>hpss_PIOSetUseAPIHost()</code>	438
9.19.2.11 <code>hpss_PIOWriteFile()</code>	438
9.19.2.12 <code>hpss_PIOWriteFileMulti()</code>	439
9.20 Trashcan	440
9.20.1 Detailed Description	440
9.20.2 Function Documentation	440
9.20.2.1 <code>hpss_GetTrashcan()</code>	440
9.20.2.2 <code>hpss_GetTrashcanHandle()</code>	441
9.20.2.3 <code>hpss_GetTrashSettings()</code>	442
9.20.2.4 <code>hpss_ReadTrash()</code>	442
9.20.2.5 <code>hpss_Undelete()</code>	443
9.20.2.6 <code>hpss_UndeleteHandle()</code>	444
9.21 Data Containers	445
9.21.1 Detailed Description	448
9.21.2 Class Documentation	449
9.21.2.1 <code>struct hpss_irbtree</code>	449
9.21.2.2 <code>struct hpss_irbtree_node</code>	450
9.21.3 Macro Definition Documentation	450
9.21.3.1 <code>hpss_irbtree_container</code>	450
9.21.4 Typedef Documentation	451
9.21.4.1 <code>hpss_ilst_alloc_func_t</code>	451
9.21.4.2 <code>hpss_ilst_destructor_t</code>	451
9.21.4.3 <code>hpss_ilst_iterator_token_h</code>	451
9.21.4.4 <code>hpss_ilst_mapped_func_t</code>	452
9.21.4.5 <code>hpss_ilst_t</code>	452
9.21.4.6 <code>hpss_irbtree_node_callback_t</code>	452
9.21.4.7 <code>hpss_irbtree_node_comparator_t</code>	453
9.21.4.8 <code>hpss_irbtree_node_destructor_t</code>	453
9.21.4.9 <code>hpss_irbtree_node_printer_t</code>	453
9.21.5 Enumeration Type Documentation	453
9.21.5.1 <code>hpss_htable_constants</code>	453
9.21.5.2 <code>hpss_irbtree_color</code>	454
9.21.6 Function Documentation	454
9.21.6.1 <code>hpss_ilst_append()</code>	454
9.21.6.2 <code>hpss_ilst_clear()</code>	454
9.21.6.3 <code>hpss_ilst_custom_init()</code>	455
9.21.6.4 <code>hpss_ilst_empty()</code>	456
9.21.6.5 <code>hpss_ilst_foreach()</code>	456
9.21.6.6 <code>hpss_ilst_foreach_range()</code>	456

9.21.6.7	hpss_ilst_get_current()	457
9.21.6.8	hpss_ilst_get_current_r()	457
9.21.6.9	hpss_ilst_get_node()	458
9.21.6.10	hpss_ilst_get_size()	458
9.21.6.11	hpss_ilst_halt_iterator()	459
9.21.6.12	hpss_ilst_init()	459
9.21.6.13	hpss_ilst_init_iterator()	459
9.21.6.14	hpss_ilst_init_iterator_r()	460
9.21.6.15	hpss_ilst_init_iterator_range()	460
9.21.6.16	hpss_ilst_init_iterator_range_r()	461
9.21.6.17	hpss_ilst_insert()	461
9.21.6.18	hpss_ilst_join()	462
9.21.6.19	hpss_ilst_prepend()	462
9.21.6.20	hpss_ilst_reinit_iterator()	463
9.21.6.21	hpss_ilst_reinit_iterator_range()	463
9.21.6.22	hpss_ilst_remove()	464
9.21.6.23	hpss_ilst_remove_safe()	464
9.21.6.24	hpss_irbtree_clear()	465
9.21.6.25	hpss_irbtree_empty()	465
9.21.6.26	hpss_irbtree_find()	465
9.21.6.27	hpss_irbtree_find_cmp()	466
9.21.6.28	hpss_irbtree_foreach()	467
9.21.6.29	hpss_irbtree_foreach_node()	467
9.21.6.30	hpss_irbtree_foreach_node_prev()	468
9.21.6.31	hpss_irbtree_init()	468
9.21.6.32	hpss_irbtree_insert()	469
9.21.6.33	hpss_irbtree_insert_multi()	469
9.21.6.34	hpss_irbtree_lower_bound()	469
9.21.6.35	hpss_irbtree_max()	470
9.21.6.36	hpss_irbtree_min()	470
9.21.6.37	hpss_irbtree_node_next()	471
9.21.6.38	hpss_irbtree_node_prev()	471
9.21.6.39	hpss_irbtree_print()	472
9.21.6.40	hpss_irbtree_remove()	472
9.21.6.41	hpss_irbtree_size()	473
9.21.6.42	hpss_irbtree_upper_bound()	473
9.22	HPSS Environment Variable Module	473
9.22.1	Detailed Description	474
9.22.2	Function Documentation	474
9.22.2.1	hpss_Envcpy()	474
9.22.2.2	hpss_EnvLocalUpdate()	474
9.22.2.3	hpss_EnvReInitialize()	475

9.22.2.4 hpss_Getenv()	475
9.23 HPSS Networking Library	475
9.23.1 Detailed Description	476
9.23.2 Function Documentation	476
9.23.2.1 hpss_net_accept()	476
9.23.2.2 hpss_net_addrmatch()	477
9.23.2.3 hpss_net_bind()	477
9.23.2.4 hpss_net_connect()	479
9.23.2.5 hpss_net_generatemask()	480
9.23.2.6 hpss_net_get_family_option()	480
9.23.2.7 hpss_net_get_family_string()	481
9.23.2.8 hpss_net_getinaddr()	481
9.23.2.9 hpss_net_getnameinfo()	482
9.23.2.10 hpss_net_getpeername()	482
9.23.2.11 hpss_net_getport()	483
9.23.2.12 hpss_net_getuniversaladdress()	483
9.23.2.13 hpss_net_initaddr()	484
9.23.2.14 hpss_net_listen()	484
9.23.2.15 hpss_net_setport()	485
9.23.2.16 hpss_net_socket()	485
9.23.2.17 hpss_net_strerror()	486
9.23.2.18 hpss_net_universaladdressstoport()	487
9.24 HPSS XML Interface	487
9.24.1 Detailed Description	488
9.24.2 Function Documentation	488
9.24.2.1 hpss_ChompXMLHeader()	488
9.24.2.2 hpss_CreateXMLWhere()	488
9.24.2.3 hpss_CreateXQueryDelete()	489
9.24.2.4 hpss_CreateXQueryGet()	490
9.24.2.5 hpss_CreateXQueryUpdate()	490
9.24.2.6 hpss_ValidXPath()	491
9.24.2.7 hpss_XMLtoAttr()	492
9.24.2.8 hpss_XPathBranch()	493
9.24.2.9 hpss_XPathLeaf()	493
9.24.2.10 hpss_XPathRBranch()	494
9.24.2.11 hpss_XPathRoot()	494
9.24.2.12 hpss_XPathtoXML()	495
9.25 64-bit Math Library	496
9.25.1 Detailed Description	497
9.25.2 Macro Definition Documentation	498
9.25.2.1 add64_3m	498
9.25.2.2 add64m	498

9.25.2.3 and64m	498
9.25.2.4 andbit64m	498
9.25.2.5 bld64m	498
9.25.2.6 cast32m	499
9.25.2.7 cast64m	499
9.25.2.8 chkbit64m	499
9.25.2.9 clrbit64m	499
9.25.2.10 CONVERT_LONGLONG_TO_U64	499
9.25.2.11 CONVERT_U64_TO_LONGLONG	499
9.25.2.12 dec64m	500
9.25.2.13 div2x64m	500
9.25.2.14 div64_3m	500
9.25.2.15 div64m	500
9.25.2.16 eq64m	500
9.25.2.17 eqz64m	500
9.25.2.18 ge64m	501
9.25.2.19 gt64m	501
9.25.2.20 high32m	501
9.25.2.21 inc64m	501
9.25.2.22 le64m	501
9.25.2.23 low32m	501
9.25.2.24 lt64m	502
9.25.2.25 mem64m	502
9.25.2.26 mod2x64m	502
9.25.2.27 mod64_3m	502
9.25.2.28 mod64m	502
9.25.2.29 mul64_3m	503
9.25.2.30 mul64m	503
9.25.2.31 neq64m	503
9.25.2.32 neqz64m	503
9.25.2.33 not64m	503
9.25.2.34 or64m	503
9.25.2.35 orbit64m	504
9.25.2.36 shl64_3m	504
9.25.2.37 shl64_ipm	504
9.25.2.38 shl64m	504
9.25.2.39 shr64_3m	504
9.25.2.40 shr64_ipm	505
9.25.2.41 shr64m	505
9.25.2.42 sub64_3m	505
9.25.2.43 sub64m	505
9.26 HPSS Client API Extensions	505

9.26.1 Detailed Description	506
9.26.2 Macro Definition Documentation	506
9.26.2.1 MIN_RESTRICTED_PORT	506
9.26.3 Function Documentation	507
9.26.3.1 hpsscfx_ConfGetClientInterfaces()	507
9.26.3.2 hpsscfx_ConfParse()	508
9.26.3.3 hpsscfx_ConfSetDirPaths()	509
9.26.3.4 hpsscfx_ConfSetFileName()	509
9.26.3.5 hpsscfx_LookupHostName()	510
9.26.3.6 hpsscfx_NetoptFindEntry()	511
9.26.3.7 hpsscfx_NetoptGetWriteSize()	511
9.26.3.8 hpsscfx_NetoptSetSock()	512
9.26.3.9 hpsscfx_PatternMatch()	512
9.27 HPSS Config File Template API	513
9.27.1 Detailed Description	513
9.28 Main	514
9.28.1 Detailed Description	514
9.28.2 Function Documentation	514
9.28.2.1 hpsscfx_DisposeConf()	514
9.28.2.2 hpsscfx_FormatMessage()	514
9.28.2.3 hpsscfx_ParseTemplate()	515
9.28.2.4 hpsscfx_ReadConf()	515
9.28.2.5 hpsscfx_ReadConfTemplate()	516
9.29 Parsers	516
9.29.1 Detailed Description	517
9.29.2 Function Documentation	517
9.29.2.1 HPSSCFGX_ENUM()	517
9.29.2.2 HPSSCFGX_STRING()	518
9.30 Special	518
9.30.1 Detailed Description	518
9.31 Validators	518
9.31.1 Detailed Description	519
9.32 POSIX APIs	519
9.32.1 Detailed Description	519
9.32.2 Function Documentation	519
9.32.2.1 hpss_CreateWithHints()	519
9.32.2.2 hpss_OpenWithHints()	520
9.32.2.3 hpss_PosixCreate()	521
9.32.2.4 hpss_PosixFstat()	521
9.32.2.5 hpss_PosixLseek()	522
9.32.2.6 hpss_PosixLstat()	522
9.32.2.7 hpss_PosixOpen()	523

9.32.2.8 hpss_PosixStat()	523
9.33 Client API Location Services	524
9.33.1 Detailed Description	524
9.33.2 Function Documentation	524
9.33.2.1 hpss_LocateRootCS()	524
9.33.2.2 hpss_LocateServerByID()	525
9.33.2.3 hpss_LocateServersByType()	526
9.33.2.4 hpss_LocateSiteByld()	526
9.33.2.5 hpss_LocateSiteByName()	527
9.34 Hash Interface	527
9.34.1 Detailed Description	529
9.34.2 Macro Definition Documentation	529
9.34.2.1 HPSS_CKSUM_AFALG	529
9.34.2.2 HPSS_CKSUM_SSE42	529
9.34.2.3 HPSS_CKSUM_VPMSUM	529
9.34.3 Function Documentation	529
9.34.3.1 hpss_HashAppend()	529
9.34.3.2 hpss_HashAppendBuf()	530
9.34.3.3 hpss_HashAppendStr()	530
9.34.3.4 hpss_HashBaselnit()	531
9.34.3.5 hpss_HashCreate()	531
9.34.3.6 hpss_HashDecode()	532
9.34.3.7 hpss_HashDelete()	532
9.34.3.8 hpss_HashDigestLength()	533
9.34.3.9 hpss_HashDigestToHex()	533
9.34.3.10 hpss_HashDigestToString()	534
9.34.3.11 hpss_HashDuplicate()	534
9.34.3.12 hpss_HashEncode()	535
9.34.3.13 hpss_HashExtract()	535
9.34.3.14 hpss_HashFinish()	536
9.34.3.15 hpss_HashFinishDigest()	536
9.34.3.16 hpss_HashFinishHex()	537
9.34.3.17 hpss_HashGetHWFlags()	537
9.34.3.18 hpss_HashLoad()	538
9.34.3.19 hpss_HashReset()	538
9.34.3.20 hpss_HashType()	539
9.34.3.21 hpss_HexStringToHashDigest()	539
9.35 HPSS Config File API	539
9.35.1 Detailed Description	542
9.35.2 Class Documentation	542
9.35.2.1 struct hpss_cfg_stanza	542
9.35.3 Macro Definition Documentation	543

9.35.3.1 HPSS_CFG_MAX_CHARS_PER_LINE	543
9.35.3.2 HPSS_CFG_MAX_LEVEL	543
9.35.4 Typedef Documentation	543
9.35.4.1 hpss_cfg_stanza_t	543
9.35.5 Enumeration Type Documentation	543
9.35.5.1 HPSS_CFG_STANZA_TYPE	543
9.35.6 Function Documentation	544
9.35.6.1 hpss_CfgFindKey()	544
9.35.6.2 hpss_CfgFindToken()	544
9.35.6.3 hpss_CfgFree()	545
9.35.6.4 hpss_CfgParse()	545
9.35.6.5 hpsscfx_ConfParse()	546
9.36 HPSS Access Control List APIs	547
9.36.1 Detailed Description	548
9.36.2 Class Documentation	548
9.36.2.1 struct hacl_acl_entry	548
9.36.3 Macro Definition Documentation	548
9.36.3.1 HACL_MAX_ENTRY_SIZE	548
9.36.3.2 HPSS_EHACL_UNAUTH	549
9.36.4 Typedef Documentation	549
9.36.4.1 hacl_acl_entry_t	549
9.36.5 Function Documentation	549
9.36.5.1 hacl_ConvertACLToHACL()	549
9.36.5.2 hacl_ConvertHACLPermsToPerms()	550
9.36.5.3 hacl_ConvertHACLToACL()	551
9.36.5.4 hacl_ConvertHACLToString()	551
9.36.5.5 hacl_ConvertHACLTypeToType()	552
9.36.5.6 hacl_ConvertPermsToHACLPerms()	553
9.36.5.7 hacl_ConvertStringsToHACL()	553
9.36.5.8 hacl_ConvertTypeToHACLType()	554
9.36.5.9 hacl_DeleteHACL()	555
9.36.5.10 hacl_GetHACL()	555
9.36.5.11 hacl_SetHACL()	555
9.36.5.12 hacl_SortHACL()	556
9.36.5.13 hacl_UpdateHACL()	556
9.37 Gatekeeper Site Library	557
9.37.1 Detailed Description	558
9.37.2 Class Documentation	558
9.37.2.1 struct gk_EntryInfo	558
9.37.2.2 struct gk_ReqInfo	559
9.37.3 Typedef Documentation	559
9.37.3.1 gk_EntryInfo_t	559

9.37.3.2 gk_ReqInfo_t	559
9.37.4 Function Documentation	559
9.37.4.1 gk_site_Close()	559
9.37.4.2 gk_site_CreateComplete()	560
9.37.4.3 gk_site_CreateStats()	561
9.37.4.4 gk_site_GetMonitorTypes()	562
9.37.4.5 gk_site_Init()	563
9.37.4.6 gk_site_Open()	564
9.37.4.7 gk_site_OpenStats()	564
9.37.4.8 gk_site_Stage()	565
9.37.4.9 gk_site_StageComplete()	565
9.37.4.10 gk_site_StageStats()	566
9.38 Account Validation Site Library	567
9.38.1 Detailed Description	567
9.38.2 Function Documentation	567
9.38.2.1 av_site_AcctIdxToName()	568
9.38.2.2 av_site_AcctNameToIdx()	569
9.38.2.3 av_site_Initialize()	570
9.38.2.4 av_site_Shutdown()	571
9.38.2.5 av_site_ValidateAccount()	571
9.38.2.6 av_site_ValidateChacct()	573
9.38.2.7 av_site_ValidateChown()	574
9.38.2.8 av_site_ValidateCreate()	575
10 File Documentation	579
10.1 acct_Site.c File Reference	579
10.2 acct_site.h File Reference	580
10.3 api_access.c File Reference	580
10.3.1 Detailed Description	581
10.4 api_acct.c File Reference	581
10.4.1 Detailed Description	582
10.4.2 Function Documentation	582
10.4.2.1 API_DetermineAcct()	582
10.4.2.2 API_GetAcctForParent()	583
10.4.2.3 API_PutAcctForParent()	583
10.4.2.4 API_SetCurrentSessionAcct()	584
10.5 api_acl.c File Reference	584
10.5.1 Detailed Description	585
10.6 api_cache.c File Reference	585
10.6.1 Detailed Description	587
10.6.2 Function Documentation	587
10.6.2.1 API_AddFSDirCache()	587

10.6.2.2 API_AddHomeDirCache()	587
10.6.2.3 API_ClearFSDirCache()	588
10.6.2.4 API_ClearHomeDirCache()	588
10.6.2.5 API_GetFSDirCache()	589
10.6.2.6 API_GetHomeDirCache()	589
10.6.2.7 API_GetJunctionCache()	590
10.6.2.8 API_InitFSDirCache()	590
10.6.2.9 API_InitHomeDirCache()	590
10.6.2.10 API_RebuildJunctionCache()	591
10.6.2.11 API_RemoveFSDirCache()	591
10.6.2.12 API_RemoveHomeDirCache()	591
10.6.2.13 fsrootkeycompare()	592
10.6.2.14 stale_thresh_cmp_factor()	592
10.6.2.15 thresh_cmp()	593
10.6.2.16 userkeycompare()	593
10.7 api_chdir.c File Reference	594
10.7.1 Detailed Description	594
10.8 api_chmod.c File Reference	594
10.8.1 Detailed Description	594
10.9 api_chown.c File Reference	595
10.9.1 Detailed Description	595
10.10 api_chroot.c File Reference	595
10.10.1 Detailed Description	595
10.10.2 Function Documentation	595
10.10.2.1 API_ApplyRootDir()	596
10.11 api_close.c File Reference	596
10.11.1 Detailed Description	596
10.12 api_closedir.c File Reference	597
10.12.1 Detailed Description	597
10.13 api_copyfile.c File Reference	597
10.13.1 Detailed Description	597
10.14 api_cos.c File Reference	597
10.14.1 Detailed Description	598
10.15 api_distfile.c File Reference	598
10.15.1 Detailed Description	598
10.16 api_fgetattr.c File Reference	598
10.16.1 Detailed Description	599
10.16.2 Function Documentation	600
10.16.2.1 API_ChecksumStateToFlags()	600
10.16.2.2 API_GetMultiFileDigest()	600
10.16.2.3 API_GetMultiFileDigestHandle()	601
10.16.2.4 API_GetSingleFileDigestHandle()	602

10.16.2.5 API_UDAFindKey()	603
10.17 api_fileset.c File Reference	603
10.17.1 Detailed Description	604
10.18 api_fsetattr.c File Reference	604
10.18.1 Detailed Description	605
10.19 api_getcwd.c File Reference	605
10.19.1 Detailed Description	605
10.20 api_gettrash.c File Reference	606
10.20.1 Detailed Description	606
10.21 api_ids.c File Reference	606
10.21.1 Detailed Description	607
10.22 api_init.c File Reference	607
10.22.1 Detailed Description	609
10.22.2 Function Documentation	609
10.22.2.1 _internal_hpss_LocateRootCS()	609
10.22.2.2 _internal_hpss_LocateServerByID()	610
10.22.2.3 _internal_hpss_LocateServersByType()	610
10.22.2.4 API_ClientAPIInit()	610
10.22.2.5 API_DebugPrintError()	611
10.22.2.6 API_DebugPrintRequest()	611
10.22.2.7 API_DebugPrintTrace()	612
10.22.2.8 API_GetServerIDFromObjHandle()	612
10.22.2.9 API_GrabConn()	613
10.22.2.10 API_InitRootHandle()	613
10.22.2.11 API_ReleaseConn()	614
10.22.2.12 API_SetAcctID()	615
10.22.2.13 API_SetCurrentGID()	615
10.22.2.14 API_SetCwd()	615
10.22.2.15 API_SetThreadCwd()	616
10.22.2.16 API_SetUmask()	616
10.22.3 Variable Documentation	617
10.22.3.1 API_TransferType	617
10.23 api_junction.c File Reference	617
10.23.1 Detailed Description	618
10.24 api_link.c File Reference	618
10.24.1 Detailed Description	618
10.25 api_lookup.c File Reference	618
10.25.1 Detailed Description	619
10.26 api_lseek.c File Reference	619
10.26.1 Detailed Description	619
10.27 api_misc.c File Reference	619
10.27.1 Detailed Description	623

10.27.2 Function Documentation	623
10.27.2.1 API_AllocateRequestIOR()	623
10.27.2.2 API_BuildFileSrcSink()	624
10.27.2.3 API_BuildXferSrcSink()	624
10.27.2.4 API_CalcBufOffsetAndSize()	625
10.27.2.5 API_CancelDataTask()	626
10.27.2.6 API_CheckFileDescBounds()	626
10.27.2.7 API_CleanupInterruptBuffer()	627
10.27.2.8 API_ClearFileTableBusy()	627
10.27.2.9 API_CloseDataConnList()	628
10.27.2.10 API_CloseListenDesc()	628
10.27.2.11 API_CloseThrdCntlDesc()	628
10.27.2.12 API_ConvertAttrToVAttrSelFlags()	629
10.27.2.13 API_ConvertVAttrsToAttrs()	629
10.27.2.14 API_ConvertVAttrToAttrSelFlags()	630
10.27.2.15 API_DescListAlloc()	630
10.27.2.16 API_DescListClear()	630
10.27.2.17 API_DescListFree()	631
10.27.2.18 API_DescListIsSet()	631
10.27.2.19 API_DescListSet()	631
10.27.2.20 API_DescListZero()	632
10.27.2.21 API_DivideFilePath()	632
10.27.2.22 API_ExpandFileTable()	632
10.27.2.23 API_FileTableSize()	633
10.27.2.24 API_FillFreeList()	633
10.27.2.25 API_FreeIODPtrs()	634
10.27.2.26 API_FreeRequestIOR()	634
10.27.2.27 API_GetAllSubsystems()	634
10.27.2.28 API_GetAppliedPath()	635
10.27.2.29 API_GetNextIORRequestID()	635
10.27.2.30 API_GetObjectFilesetId()	636
10.27.2.31 API_GetOpenTableSlot()	637
10.27.2.32 API_HexDecode()	637
10.27.2.33 API_HexEncode()	637
10.27.2.34 API_InitializeServerIOD()	638
10.27.2.35 API_InsertInFreeList()	638
10.27.2.36 API_InterruptDataTask()	639
10.27.2.37 API_LockCondition()	639
10.27.2.38 API_LockMutex()	640
10.27.2.39 API_net_bind()	640
10.27.2.40 API_Net_CreateAddr_INADDR_ANY()	640
10.27.2.41 API_ObjHandleToString()	641

10.27.2.42 API_OpenListenDesc()	641
10.27.2.43 API_OpenThrdCntlDesc()	642
10.27.2.44 API_PerformRetry()	642
10.27.2.45 API_RemoveCachedTrash()	644
10.27.2.46 API_RetrieveTrashcanHandle()	644
10.27.2.47 API_SetFileTableBusy()	645
10.27.2.48 API_SetLastHPSSErrno()	646
10.27.2.49 API_TranslateError()	646
10.27.2.50 API_UnlockCondition()	647
10.27.2.51 API_UnlockFileTableEntries()	647
10.27.2.52 API_UnlockMutex()	648
10.28 api_misc_admin.c File Reference	648
10.28.1 Detailed Description	648
10.29 api_mkdir.c File Reference	648
10.29.1 Detailed Description	649
10.30 api_open.c File Reference	649
10.30.1 Detailed Description	650
10.31 api_opendir.c File Reference	650
10.31.1 Detailed Description	650
10.32 api_path.c File Reference	650
10.32.1 Detailed Description	651
10.32.2 Function Documentation	651
10.32.2.1 API_FreeCwdStack()	652
10.32.2.2 API_InitCwdStack()	652
10.32.2.3 API_PathFromCwdStack()	652
10.32.2.4 API_SetCwdStack()	653
10.32.2.5 API_TraversePath()	653
10.32.2.6 NormalizePath()	656
10.33 api_pio.c File Reference	656
10.33.1 Detailed Description	658
10.33.2 Function Documentation	658
10.33.2.1 PIOInit()	658
10.34 api_posix_lseek.c File Reference	658
10.35 api_posix_open.c File Reference	659
10.36 api_purge.c File Reference	659
10.36.1 Detailed Description	660
10.37 api_purgelock.c File Reference	660
10.37.1 Detailed Description	660
10.38 api_purgeonmigrate.c File Reference	660
10.38.1 Detailed Description	661
10.39 api_rddir.c File Reference	661
10.39.1 Detailed Description	662

10.39.2 Function Documentation	662
10.39.2.1 hpss_Rumble()	662
10.40 api_rdlink.c File Reference	662
10.40.1 Detailed Description	663
10.41 api_rdlst.c File Reference	663
10.41.1 Detailed Description	663
10.42 api_read.c File Reference	664
10.42.1 Detailed Description	664
10.43 api_register.c File Reference	665
10.43.1 Detailed Description	665
10.44 api_rename.c File Reference	665
10.44.1 Detailed Description	665
10.45 api_rewdir.c File Reference	666
10.45.1 Detailed Description	666
10.46 api_rmdir.c File Reference	666
10.46.1 Detailed Description	667
10.46.2 Function Documentation	667
10.46.2.1 Common_hpss_Rmdir()	667
10.46.2.2 Common_hpss_RmdirHandle()	667
10.47 api_stage.c File Reference	669
10.47.1 Detailed Description	670
10.47.2 Function Documentation	670
10.47.2.1 API_CopyBatchByStatus()	670
10.47.2.2 API_UpdateBatchStatus()	671
10.48 api_stat.c File Reference	671
10.48.1 Detailed Description	672
10.49 api_statfs.c File Reference	672
10.49.1 Detailed Description	672
10.50 api_statsubsys.c File Reference	672
10.50.1 Detailed Description	673
10.51 api_stream.c File Reference	673
10.51.1 Detailed Description	673
10.52 api_symlink.c File Reference	674
10.52.1 Detailed Description	674
10.53 api_trunc.c File Reference	674
10.53.1 Detailed Description	675
10.54 api_umask.c File Reference	675
10.54.1 Detailed Description	675
10.55 api_undelete.c File Reference	675
10.55.1 Detailed Description	676
10.56 api_unlink.c File Reference	676
10.56.1 Detailed Description	676

10.56.2 Function Documentation	676
10.56.2.1 Common_hpss_Unlink()	676
10.57 api_userattr.c File Reference	677
10.57.1 Detailed Description	678
10.58 api_utime.c File Reference	678
10.58.1 Detailed Description	679
10.59 api_wrappers.c File Reference	679
10.59.1 Detailed Description	679
10.59.2 Function Documentation	679
10.59.2.1 API_core_OpenBitfile()	680
10.60 api_write.c File Reference	680
10.60.1 Detailed Description	681
10.61 api_wrtlist.c File Reference	681
10.61.1 Detailed Description	681
10.62 gk_RTM.h File Reference	682
10.63 gk_SiteProtos.h File Reference	682
10.64 hacl.h File Reference	682
10.65 hpss_api.h File Reference	684
10.65.1 Class Documentation	700
10.65.1.1 struct api_dist_file_info_t::file_info	700
10.66 hpss_dirent.h File Reference	700
10.66.1 Detailed Description	701
10.66.2 Class Documentation	701
10.66.2.1 struct hpss_dirent	701
10.67 hpss_Getenv.c File Reference	701
10.67.1 Detailed Description	702
10.67.2 Function Documentation	702
10.67.2.1 hpss_GetenvDefault()	702
10.68 hpss_Getenv.h File Reference	702
10.68.1 Function Documentation	703
10.68.1.1 hpss_GetenvDefault()	703
10.69 hpss_hash.c File Reference	703
10.69.1 Detailed Description	705
10.69.2 Macro Definition Documentation	705
10.69.2.1 ADLER32_BASE	705
10.70 hpss_hash.h File Reference	705
10.71 hpss_ihashtable.h File Reference	706
10.71.1 Detailed Description	707
10.71.2 Typedef Documentation	707
10.71.2.1 hpss_ihashtable_node_t	707
10.72 hpss_ilist.h File Reference	707
10.72.1 Detailed Description	709

10.72.2 Typedef Documentation	710
10.72.2.1 hpss_ilst_foreach_retval_e	710
10.72.3 Enumeration Type Documentation	711
10.72.3.1 hpss_ilst_foreach_retval	711
10.73 hpss_irbtree.c File Reference	711
10.73.1 Detailed Description	713
10.74 hpss_irbtree.h File Reference	713
10.74.1 Detailed Description	715
10.75 hpss_Mech.c File Reference	715
10.75.1 Detailed Description	716
10.75.2 Function Documentation	716
10.75.2.1 hpss_GetAuthType()	716
10.76 hpss_net.c File Reference	716
10.76.1 Detailed Description	718
10.76.2 Function Documentation	718
10.76.2.1 __hpss_net_maskaddr32()	718
10.76.2.2 __hpss_net_setscopeid()	718
10.76.2.3 hpss_net_getaddrinfo()	719
10.76.2.4 hpss_net_getaddrinfo_ex()	720
10.76.2.5 hpss_net_getsockname()	721
10.77 hpss_net.h File Reference	721
10.77.1 Function Documentation	723
10.77.1.1 __hpss_net_maskaddr32()	723
10.77.1.2 __hpss_net_setscopeid()	723
10.77.1.3 hpss_net_getaddrinfo()	724
10.77.1.4 hpss_net_getaddrinfo_ex()	725
10.77.1.5 hpss_net_getsockname()	726
10.78 hpss_posix_api.h File Reference	726
10.78.1 Detailed Description	727
10.79 hpss_stat.h File Reference	727
10.80 hpss_String.c File Reference	728
10.80.1 Detailed Description	731
10.80.2 Function Documentation	731
10.80.2.1 hpss_AdminStateString()	731
10.80.2.2 hpss_AllocStateString()	732
10.80.2.3 hpss_AuthenticatorTypeFromString()	732
10.80.2.4 hpss_AuthenticatorTypeString()	732
10.80.2.5 hpss_AuthnMechTypeFromString()	733
10.80.2.6 hpss_AuthnMechTypeString()	733
10.80.2.7 hpss_AuthzMechTypeFromString()	733
10.80.2.8 hpss_AuthzMechTypeString()	734
10.80.2.9 hpss_Bytcount32()	734

10.80.2.10 hpss_Bytecount64()	735
10.80.2.11 hpss_DriveFlagString()	735
10.80.2.12 hpss_DriveStateString()	735
10.80.2.13 hpss_DriveTypeString()	736
10.80.2.14 hpss_ErrnoName()	736
10.80.2.15 hpss_getline()	736
10.80.2.16 hpss_HashTypeString()	737
10.80.2.17 hpss_HexString()	737
10.80.2.18 hpss_LabelTypeString()	738
10.80.2.19 hpss_MediaTypeString()	738
10.80.2.20 hpss_MountModeString()	738
10.80.2.21 hpss_OpStateString()	739
10.80.2.22 hpss_ProtLevelFromString()	739
10.80.2.23 hpss_ProtLevelString()	740
10.80.2.24 hpss_PVLActivityStateString()	740
10.80.2.25 hpss_PVLJobStateString()	740
10.80.2.26 hpss_PVRTypeString()	741
10.80.2.27 hpss_ServerTypeString()	741
10.80.2.28 hpss_string_cmp()	741
10.80.2.29 hpss_string_fromCstr()	742
10.80.2.30 hpss_stringbuf_clear()	742
10.80.2.31 hpss_stringbuf_cmp()	742
10.80.2.32 hpss_stringbuf_delBackward()	743
10.80.2.33 hpss_stringbuf_delForward()	743
10.80.2.34 hpss_stringbuf_free()	744
10.80.2.35 hpss_stringbuf_fromCstr()	744
10.80.2.36 hpss_stringbuf_fromFile()	745
10.80.2.37 hpss_stringbuf_fromString()	745
10.80.2.38 hpss_stringbuf_grow()	746
10.80.2.39 hpss_stringbuf_insert()	746
10.80.2.40 hpss_stringbuf_printf()	747
10.80.2.41 hpss_StringClean()	747
10.80.2.42 hpss_StringsAreEqual()	748
10.80.2.43 hpss_time_string()	748
10.80.2.44 hpss_Token_Add()	749
10.80.2.45 hpss_Tokenize()	749
10.80.2.46 hpss_UsageStateString()	749
10.80.2.47 s32_and_map()	750
10.80.2.48 s32_eq_map()	750
10.80.2.49 u32_and_map()	751
10.80.2.50 u32_eq_map()	751
10.80.3 Variable Documentation	751

10.80.3.1 AdminStateStrings	752
10.80.3.2 AuthenticatorTypeStrings	752
10.80.3.3 AuthnMechStrings	752
10.80.3.4 AuthzMechStrings	752
10.80.3.5 DriveFlagStrings	753
10.80.3.6 DriveStateStrings	753
10.80.3.7 LabelTypeStrings	753
10.80.3.8 NumAdminStateStrings	753
10.80.3.9 NumAuthenticatorTypeStrings	754
10.80.3.10 NumAuthnMechStrings	754
10.80.3.11 NumAuthzMechStrings	754
10.80.3.12 NumDriveFlagStrings	754
10.80.3.13 NumDriveStateStrings	754
10.80.3.14 NumErrnoStrings	755
10.80.3.15 NumLabelTypeStrings	755
10.80.3.16 NumOpStateStrings	755
10.80.3.17 NumProtLevelStrings	755
10.80.3.18 NumPVLActivityStateStrings	755
10.80.3.19 NumPVLJobStateStrings	756
10.80.3.20 NumPVRTTypeStrings	756
10.80.3.21 NumServerSubtypeStrings	756
10.80.3.22 NumServerTypeStrings	756
10.80.3.23 NumUsageStateStrings	756
10.80.3.24 OpStateStrings	757
10.80.3.25 ProtLevelStrings	757
10.80.3.26 PVLActivityStateStrings	757
10.80.3.27 PVLJobStateStrings	758
10.80.3.28 PVRTTypeStrings	758
10.80.3.29 ServerSubtypeStrings	758
10.80.3.30 ServerTypeStrings	759
10.80.3.31 UsageStateStrings	759
10.81 hpss_vattr.h File Reference	759
10.81.1 Detailed Description	759
10.82 hpss_xml.c File Reference	760
10.82.1 Detailed Description	760
10.83 hpss_xml.h File Reference	761
10.84 hpsscfg_config_api.c File Reference	761
10.84.1 Detailed Description	762
10.85 hpsscfgx_cfg_functions.c File Reference	762
10.85.1 Detailed Description	762
10.86 hpsscfgx_GetClientInterfaces.c File Reference	763
10.86.1 Detailed Description	763

10.87 hpsscfgx_hpssconf.c File Reference	763
10.87.1 Detailed Description	764
10.88 hpsscfgx_pattern_match.c File Reference	764
10.88.1 Detailed Description	765
10.88.2 Macro Definition Documentation	765
10.88.2.1 MAX_PATTERN_BUF	765
10.89 hpsscfgx_restricted_ports.c File Reference	765
10.89.1 Detailed Description	766
10.90 u_signed64.h File Reference	766
10.90.1 Macro Definition Documentation	767
10.90.1.1 CS_DEC64_LEN	767
Index	769

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

Copyright © 1992-2018 International Business Machines Corporation, The Regents of the University of California, Los Alamos National Security, LLC, Lawrence Livermore National Security, LLC, Sandia Corporation, and UT-Battelle.

All rights reserved.

Portions of this work were produced by Lawrence Livermore National Security, LLC, Lawrence Livermore National Laboratory (LLNL) under Contract No. DE-AC52-07NA27344 with the U.S. Department of Energy (DOE); by the University of California, Lawrence Berkeley National Laboratory (LBNL) under Contract No. DE-AC02-05CH11231 with DOE; by Los Alamos National Security, LLC, Los Alamos National Laboratory (LANL) under Contract No. DE-AC52-06NA25396 with DOE; by Sandia Corporation, Sandia National Laboratories (SNL) under Contract No. DE-AC04-94AL85000 with DOE; and by UT-Battelle, Oak Ridge National Laboratory (ORNL) under Contract No. DE-AC05-00OR22725 with DOE. The U.S. Government has certain reserved rights under its prime contracts with the Laboratories. DISCLAIMER

Portions of this software were sponsored by an agency of the United States Government. Neither the United States, DOE, The Regents of the University of California, Los Alamos National Security, LLC, Lawrence Livermore National Security, LLC, Sandia Corporation, UT-Battelle, nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

2.2 Trademark usage

High Performance Storage System is a trademark of International Business Machines Corporation.

IBM is a registered trademark of International Business Machines Corporation.

IBM, DB2, DB2 Universal Database, AIX, RISC/6000, pSeries, and xSeries are trademarks or registered trademarks of International Business Machines Corporation.

UNIX is a registered trademark of the Open Group.

Linux is a registered trademark of Linus Torvalds in the United States and other countries.

Kerberos is a trademark of the Massachusetts Institute of Technology.

Java is a registered trademark of Oracle and/or its affiliates.

ACSL is a trademark of Oracle and/or its affiliates.

Microsoft Windows is a registered trademark of Microsoft Corporation.

Other brands and product names appearing herein may be trademarks or registered trademarks of third parties.

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](#)
- [Simplified 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 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.

PIO is a highly flexible I/O model that supports striping and custom I/O handling functions within the application - it even supports having participants running on a set of remote nodes. This flexibility can be highly useful in some cases and certain types of applications. A simplified interface is provided to handle the scenario of a local file transfer involving HPSS (see [Simplified Parallel I/O](#)).

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 PIO Simplified APIs

The PIO simplified APIs ([Simplified Parallel I/O](#)) provide wrappers around the PIO functions that provide a one call, "read" and "write" interface. These functions are limited to interacting with an open file descriptor on the same host and same process. However, these APIs should be used as an alternative to a typical `hpss_Read` or `hpss_Write` operation involving local files as they will exhibit better performance with large files. Furthermore, PIO can take advantage of client striping for improved network transfer rates.

3.1.9 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.

3.1.9.1 Getting Information on Trashcans

Information about whether trashcanning is on or not, and additional settings, can be obtained by privileged 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 information.

3.1.9.2 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 any HPSS Core Server, retrieving the information needed to contact the Core Server associated with the client request, and then issuing requests to that Core Server. Each Core Server, when contacted for location information, reports the location of each Core Server and Gatekeeper server. 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.

```
% 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 Admin 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 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_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 Servers. The effects of these values are documented in the HPSS Admin 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 Core Server is 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 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 system 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:

```
#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 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 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 file 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.SCAAttrib[i].StripeWidth);
    printf("Level %d Flags: %d\n", i, attrs.SCAAttrib[i].Flags);
    printf("Level %d VVs: %d\n", i, attrs.SCAAttrib[i].NumberOfVVs);
    for(j=0; j<attrs.SCAAttrib[i].NumberOfVVs; j++)
    {

```

```

printf("VV %d RelativePosition: %d\n", j,
      attrs.SCAAttrib[i].VVAttrib[j].RelPosition);
printf("VV %d BytesonVV: %s\n", j,
      u64tostr_r(attrs.SCAAttrib[i].VVAttrib[j].BytesOnVV, buf));
if (attrs.SCAAttrib[i].VVAttrib[j].PVList != NULL)
{
    for(k=0; k < attrs.SCAAttrib[i].VVAttrib[j].PVList->List.List_len; k++)
    {
        printf("PV Name %d: %s\n", k, attrs.SCAAttrib[i].VVAttrib[j].PVList->List.List_val[k].Name);
    }
    free(attrs.SCAAttrib[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.CIntStripeWidth = 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. CIntStripeWidth 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 HPSS_PIO_TCPIP or HPSS_PIO_MVR_SELECT. HPSS_PIO_MVR_SELECT is used for SAN3P, mainly.

For non-stripe files and volumes, multiple participants and network connections can be used for the data transfer by setting CIntStripeWidth to > 1 and setting the transport to HPSS_PIO_TCPIP.

There are several options which can be OR'd together and stored in Options. They are HPSS_PIO_PUSH, HPSS_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)&&lt;64m(currentOffset, pio->size)))
{
    U64_TO_U32(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_PIOStart()` 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_PIORegister()` 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 Admin 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 "XPath" 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 == -ENOATTR)
        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_UserAttrListAttrs()` comes in. `hpss_UserAttrListAttrs()` 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 == -ENOATTR)
        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 cursoid;
memset(&cursoid, 0x0, sizeof(cursoid));
login();
subsys=atoi(argv[1]);
xquery = "$DOC/hpss";
rc = hpss_UserAttrGetObjQuery(subsys, bufsize, &ObjectList,
                             xquery, &cursoid);

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 check-summed 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 <code>xm-laddindex</code> 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+
- 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 (ACSL) 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 (HPSS)," 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

ACSL 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 that 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.

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

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 provides a 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](#) [128]

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_GetBatchAsynchStatus](#) (hpss_reqid_t CallBackId, hpss_stage_bitfile_list_t *BFObjs, hpss_stage_status_type_t Type, hpss_stage_batch_status_t *Status)

This function will be removed in a future version of HPSS. Use [hpss_GetCallbackStatus\(\)](#).

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](#).

Module [math](#)

Chapter 9

Module Documentation

9.1 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.
- [Simplified Parallel I/O](#)
Functions which wrap PIO for ease of use with local file descriptors.
- [Trashcan](#)
Functions which use the HPSS trashcan feature.
- [Data Containers](#)
Functions which implement data containers.
- [POSIX APIs](#)
Posix Compliant HPSS APIs (Unsupported)
- [Client API Location Services](#)
Functions used by client applications to locate servers.

Classes

- struct [api_config](#)
Controls optional features of the Client API configuration.
- struct [api_dist_file_info_t](#)
Distributed file information structure.
- struct [api_namespec](#)
Name specifier information.
- struct [hpss_errno_state](#)
HPSS Error Code State.
- struct [hpss_file_hash_digest](#)
Defined the input/output structure for hpss_FgetFileDigest and hpss_FsetFileDigest and hpss_FgetFileDigestList and hpss_FsetFileDigestList.
- struct [hpss_file_hash_digest_list](#)
A list of hash digests for single or multi stripe digest handling.
- struct [hpss_global_fsent_t](#)
Global fileset entry.
- struct [hpss_junction_ent_t](#)
HPSS junction entry Structure to hold junction information.
- struct [hpss_pio_gapinfo_s](#)
PIO sparse file gap information.
- struct [hpss_pio_params_s](#)
PIO I/O parameters.
- struct [hpss_stat](#)
Stat structure for HPSS.
- struct [hpss_statfs](#)
File system stats.
- struct [hpss_statfs64](#)
File system stats (64bit)
- struct [hpss_statvfs](#)
Virtual File System stat structure.
- struct [hpss_statvfs64](#)
Virtual File System stat structure (64bit)

- struct [hpss_userattr](#)
User-defined Attribute pair.
- struct [hpss_userattr_list](#)
List of User-defined attribute pairs.
- struct [subsys_stats](#)
HPSS Subsystem statistics.

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](#)
Referencing 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_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 Subsystem statistics.

Enumerations

- enum [hpss_file_hash_flags](#) {
[HPSS_FILE_HASH_DIGEST_VALID](#) = (1<<0) , [HPSS_FILE_HASH_SET_BY_USER](#) = (1<<1) ,
[HPSS_FILE_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](#) }
Referencing flags for digest list operations.
- enum [hpss_pio_operation_t](#){ [HPSS_PIO_NONE](#) , [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) , `HPSS_READDIR_RAW` = (1 << 1) , `HPSS_READDIR_NFS` = (1 << 2) }
- enum `hpss_undelete_flags_t` { `HPSS_UNDELETE_NONE` = 0 , `HPSS_UNDELETE_RESTORE_TIME` , `HPSS_UNDELETE_OVERWRITE` , `HPSS_UNDELETE_OVERWRITE_AND_RESTORE` }
- enum `namespec_type_t` { `NAMESPEC_SKIP` , `NAMESPEC_REALM` , `NAMESPEC_USER` , `NAMESPEC_GROUP` }
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_PURGE_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_ConvertAttrsToVAttrs` (const `hpss_Attrs_t` *NSAttrs, `hpss_vattr_t` *VAttrs)
Converts HPSS attributes to VFS attributes.
- `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_FreelOR` (`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 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 hpss_AbortIOByMoverId (int32_t SubsystemId, hpss_svr_id_t MoverId)`
Abort all I/O requests for the specified mover ID.
- `int hpss_AbortIOByMoverName (int32_t SubsystemId, char *MoverName)`
Abort all I/O requests for the specified mover name.
- `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.
- `char * hpss_BuildLevelString (void)`
Construct a string containing HPSS version information.
- `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_ClearLastHPSS_errno](#) ()
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.
- char ** [hpss_CopyrightStrings](#) (void)
Read the copyright file and return the strings.
- 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_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 *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_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 *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 SelfFlags, 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 SelfFlags, 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 SelfFlags, 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 *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 *Name, 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.
- [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](#) *DigestList)
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 *BFObjs, hpss_stage_status_type_t Type, hpss_stage_batch_status_t *Status)
Check the status of files in a batch request.
- int [hpss_GetCallbackStatus](#) (int SubsystemId, hpss_reqid_t CallBackId, hpss_cb_status_t *Status)
Check the status of files in a stage callback 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_GetdentsAttr](#) (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 *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_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_GetJunctionAttr](#) (const char *Path, [hpss_Attr_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)

Retrieves the junctions that are managed by the Core Server.
- [hpss_errno_state_t](#) [hpss_GetLastHPSS_errno](#) ()

Retrieves the error state of the last HPSS error.
- int [hpss_GetListAttr](#) (const char *Path, [hpss_Attr_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_ObjHandle_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.
- int32_t [hpss_LocateRootCS](#) (hpss_reqid_t RequestId, ls_map_t *Location)
Find the local root Core Server.

- `int32_t hpss_LocateServerByID` (`hpss_reqid_t RequestId`, `hpss_srvr_id_t ServerId`, `ls_map_t *Location`)
Locate a server by its Id This routine locates a server by its unique ID. If ServerId is zero (Invalid), information about the local root Core Server is returned.
- `int32_t hpss_LocateSiteById` (`hpss_reqid_t RequestId`, `hpss_id_t SiteId`, `char *SiteName`)
Lookup a site's name by its id.
- `int32_t hpss_LocateSiteByName` (`hpss_reqid_t RequestId`, `char *SiteName`, `hpss_id_t *SiteId`)
Lookup a site's id by its name.
- `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_OpenBitfileVAttr` (`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 *AttrOut`)
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_PIOInitCtx` (`hpss_pio_t` *Ctx)
Start a simplified PIO context.
- `int hpss_PIOReadFile` (int `hpss_fd`, `uint64_t` data_length, int local_fd, `uint64_t` *bytes_copied, `hpss_read_queue_item_t` *item)
Read a file from HPSS to a local file using PIO.
- `int hpss_PIOReadFileMulti` (`hpss_pio_t` *Ctx, int `hpss_fd`, `uint64_t` data_length, int local_fd, `uint64_t` *bytes_copied, `hpss_read_queue_item_t` *item)
Issue a simplified PIO read.
- `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_PIOResetCtx` (`hpss_pio_t` *Ctx)
Stop a simplified PIO context.
- `int hpss_PIOSetBlockSize` (`hpss_pio_t` *Ctx, `uint32_t` BlockSize)
Set the client transfer block size.
- `int hpss_PIOSetDefaultCtx` (`hpss_pio_t` *Ctx)
Set default values in the PIO context.
- `int hpss_PIOSetHandleGaps` (`hpss_pio_t` *Ctx, int HandleGaps)
Set the client gap handling behavior.
- `int hpss_PIOSetStripeWidth` (`hpss_pio_t` *Ctx, `uint32_t` Width)
Set the client stripe width.
- `int hpss_PIOSetUseAPIHost` (`hpss_pio_t` *Ctx, int UseAPIHost)
Set the client data hostname behavior.
- `int hpss_PIOStart` (`hpss_pio_params_t` *InputParams, `hpss_pio_grp_t` *StripeGroup)
Start a new parallel I/O group context.
- `int hpss_PIOWriteFile` (int `hpss_fd`, `uint64_t` data_length, int local_fd, `uint64_t` *bytes_copied)
Write a file to HPSS from a local file using PIO.
- `int hpss_PIOWriteFileMulti` (`hpss_pio_t` *Ctx, int `hpss_fd`, `uint64_t` data_length, int local_fd, `uint64_t` *bytes_copied)
Write a file to HPSS from a local file using PIO.
- `signed32 hpss_PreferedReadOrder` (`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, `purge_lock_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 *UserCredsP, 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.
- 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_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 SelfFlagsIn, const [hpss_vattr_t](#) *AttrIn, uint64_t *SelfFlagsOut, [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_GetUniqueRequestID](#) or from [hpss_ReadQueueAdd](#).
- int [hpss_SiteldToName](#) (const hpss_id_t *Siteld, char *SiteName)

Convert Site ID to Site Name.
- int [hpss_SiteNameToId](#) (const char *SiteName, hpss_id_t *Siteld)

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 *BFObjList, 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 *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 *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.

9.1.1 Detailed Description

Definitions for the HPSS Client API.

9.1.2 Class Documentation

9.1.2.1 struct api_config

Controls optional features of the Client API configuration.

Class Members

char	Application[256]	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[1024]	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[128]	Name to use when generating HPSS log messages Deprecated Remove in next major release.

Class Members

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 realm. 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[64]	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.

Class Members

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

9.1.2.2 struct api_dist_file_info_t

Distributed file information structure.

Class Members

uchar	CkSum	checksum digest
struct file_info	Info	Distributed file information

9.1.2.3 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[128]	Name of principal
uint32_t	RealmId	HPSS Realm Id where principal resides
char	RealmName[256]	Name of realm where principal resides
namespec_type_t	Type	Type of name represented

9.1.2.4 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[1024]	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

9.1.2.5 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[64]	Hash digest
char	Creator[16]	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

9.1.2.6 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 be 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

9.1.2.7 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[128]	Fileset name Fileset name

9.1.2.8 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	JunctionPathName[1024]	Junction path

9.1.2.9 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

9.1.2.10 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 Note that this value impacts client network parallelism and should be matched with the throughput of the backend disk storage device if possible.
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

9.1.2.11 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

Class Members

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)

9.1.2.12 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

9.1.2.13 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

Class Members

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

9.1.2.14 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

9.1.2.15 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

9.1.2.16 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

9.1.2.17 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

9.1.2.18 struct subsys_stats

HPSS Subsystem statistics.

Class Members

uint32_t	COSChangeThreadCount	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

9.1.3 Macro Definition Documentation

9.1.3.1 API_CONFIG_ALL_FLAGS

```
#define API_CONFIG_ALL_FLAGS (0x0000001F)
```

All valid flags

9.1.3.2 API_CONFIG_DEFAULT

```
#define API_CONFIG_DEFAULT (0x00000000)
```

Default config flags

9.1.3.3 API_DEBUG_ERROR

```
#define API_DEBUG_ERROR (1)
```

Error code level debugging

9.1.3.4 API_DEBUG_NONE

```
#define API_DEBUG_NONE (0)
```

No debugging

9.1.3.5 API_DEBUG_REQUEST

```
#define API_DEBUG_REQUEST (2)
```

Request level debugging

9.1.3.6 API_DEBUG_TRACE

```
#define API_DEBUG_TRACE (4)
```

Trace level debugging

9.1.3.7 API_DISABLE_CROSS_REALM

```
#define API_DISABLE_CROSS_REALM (0x00000002)
```

Disable cross-realm (defunct)

9.1.3.8 API_DISABLE_JUNCTIONS

```
#define API_DISABLE_JUNCTIONS (0x00000004)
```

Disable junction traversal

9.1.3.9 API_ENABLE_LOGGING

```
#define API_ENABLE_LOGGING (0x00000001)
```

Enable HPSS logging

9.1.3.10 API_MSGTYPE_MVRPROT

```
#define API_MSGTYPE_MVRPROT MVRPROTOCOL_MSG_TYPE
```

Mover protocol type

9.1.3.11 API_MSGTYPE_PDATA

```
#define API_MSGTYPE_PDATA PDATA_MSG_TYPE
```

PDATA protocol type

9.1.3.12 API_TRANSFER_MVRSELECT

```
#define API_TRANSFER_MVRSELECT 1
```

Mover selects the protocol

9.1.3.13 API_TRANSFER_TCP

```
#define API_TRANSFER_TCP 0
```

Use the TCP PDATA protocol

9.1.3.14 API_USE_CONFIG

```
#define API_USE_CONFIG (0x00000008)
```

Use the provided API configuration

9.1.3.15 API_USE_SAN3P

```
#define API_USE_SAN3P (0x00000010)
```

Enable SAN3P transfers

9.1.3.16 HPSS_MIGRATE_FORCE

```
#define HPSS_MIGRATE_FORCE (BFS_MIGRATE_FORCE)
```

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

9.1.3.17 HPSS_MIGRATE_HPSS_ONLY

```
#define HPSS_MIGRATE_HPSS_ONLY (0x1)
```

If on, don't migrate dmap

9.1.3.18 HPSS_MIGRATE_NO_COPY

```
#define HPSS_MIGRATE_NO_COPY (BFS_MIGRATE_NO_COPY)
```

If on, do not migrate multiple copies

9.1.3.19 HPSS_MIGRATE_PURGE_DATA

```
#define HPSS_MIGRATE_PURGE_DATA (BFS_MIGRATE_PURGE_DATA)
```

If on, purge data after migration is complete

9.1.3.20 HPSS_O_EXCL

```
#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.

9.1.3.21 HPSS_O_NO_TAPE

```
#define HPSS_O_NO_TAPE (0x10000000)
```

Do not use tape

9.1.3.22 HPSS_O_STAGE_ASYNC

```
#define HPSS_O_STAGE_ASYNC (0x80000000)
```

Stage asynchronously on open

9.1.3.23 HPSS_O_STAGE_BKGRD

```
#define HPSS_O_STAGE_BKGRD (0x40000000)
```

Stage in the background

9.1.3.24 HPSS_O_STAGE_NONE

```
#define HPSS_O_STAGE_NONE (O_NONBLOCK)
```

Do not stage on open

9.1.3.25 HPSS_PIO_HANDLE_GAP

```
#define HPSS_PIO_HANDLE_GAP (1<<1)
```

PIO will handle file gaps

9.1.3.26 HPSS_PIO_PORT_RANGE

```
#define HPSS_PIO_PORT_RANGE (1<<2)
```

Use specified port range

9.1.3.27 HPSS_PIO_PUSH

```
#define HPSS_PIO_PUSH (1<<0)
```

Use the PDATA PUSH protocol

9.1.3.28 HPSS_PIO_USE_API_HOSTNAME

```
#define HPSS_PIO_USE_API_HOSTNAME (1<<3)
```

Listen on the API_HOSTNAME interface by default

9.1.3.29 HPSS_STAGE_STATUS_ACTIVE

```
#define HPSS_STAGE_STATUS_ACTIVE (2)
```

Stage request is being processed

9.1.3.30 HPSS_STAGE_STATUS_QUEUED

```
#define HPSS_STAGE_STATUS_QUEUED (1)
```

Stage request is queued in the bitfile layer

9.1.3.31 HPSS_STAGE_STATUS_UNKNOWN

```
#define HPSS_STAGE_STATUS_UNKNOWN (0)
```

Stage state is unknown

9.1.3.32 MAX_XPATH_ELEM_SIZE

```
#define MAX_XPATH_ELEM_SIZE (1001)
```

Maximum size of an XML element

9.1.3.33 MAX_XPATH_SIZE

```
#define MAX_XPATH_SIZE (MAX_XPATH_ELEM_SIZE*125)
```

Maximum XPath size

9.1.3.34 UDA_API_VALUE

```
#define UDA_API_VALUE 0
```

Provide the text value only

9.1.3.35 UDA_API_XML

```
#define UDA_API_XML 1
```

Provide the ML

9.1.3.36 XML_ATTR

```
#define XML_ATTR 1
```

Include XML attribute fields

9.1.3.37 XML_NO_ATTR

```
#define XML_NO_ATTR 2
```

Do not include XML attribute fields

9.1.4 Typedef Documentation

9.1.4.1 api_namespec_t

```
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.

9.1.4.2 hpss_errno_state_t

```
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.

9.1.4.3 HPSS_FILE

```
typedef struct __HPSS_FILE HPSS_FILE
```

HPSS file stream

9.1.4.4 hpss_file_hash_flags_t

```
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)

9.1.4.5 hpss_pio_options_t

```
typedef uint32_t hpss_pio_options_t
```

Parallel IO transport options

9.1.4.6 hpss_userattr_t

```
typedef struct hpss_userattr hpss_userattr_t
```

User-defined Attribute pair.

Prototypes and Data Structures for User-defined Attributes

9.1.5 Enumeration Type Documentation

9.1.5.1 hpss_file_hash_flags

```
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

9.1.5.2 hpss_file_hash_stripe_flags

```
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

9.1.5.3 hpss_pio_operation_t

```
enum hpss_pio_operation_t
```

Parallel IO operation types

Enumerator

HPSS_PIO_NONE	No operation (invalid)
HPSS_PIO_READ	Read Operation
HPSS_PIO_WRITE	Write Operation

9.1.5.4 hpss_pio_transport_t

```
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

9.1.5.5 hpss_readdir_flags_t

```
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

9.1.5.6 hpss_undelete_flags_t

```
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

9.1.5.7 namespec_type_t

```
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_REALM_AND_USER	Translate realm name and user name

9.1.5.8 notrunc_flag_t

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

9.1.5.9 purgelock_flag_t

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)

9.1.5.10 purgeonmigrate_flag_t

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

9.1.6 Function Documentation

9.1.6.1 API_AddAllRegisterValues()

```
uint64_t API_AddAllRegisterValues (
    int LastPosition )
```

Set all register bit positions up to the position provided.

Parameters

in	<i>LastPosition</i>	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](#)

9.1.6.2 API_AddRegisterValues()

```
uint64_t API_AddRegisterValues (
    uint64_t InitialValue,
    ... )
```

Add values to a bit register.

Parameters

in	<i>InitialValue</i>	Initial value
in	...	A series of bit positions to turn on within the provided <i>InitialValue</i> . 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](#)

9.1.6.3 API_CalcBlock32()

```
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	<i>Attrs</i>	HPSS object attributes
out	<i>Blksize</i>	Block Size
out	<i>Numblks</i>	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.

9.1.6.4 API_CalcBlock64()

```
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	<i>Attrs</i>	HPSS object attributes
out	<i>Blksize</i>	Block Size
out	<i>Numblks</i>	Number of Blocks

See also

[API_CalcBlock32](#), [API_ConvertModeToPosixMode](#), [API_ConvertPosixModeToMode](#), [API_ConvertPosixTimeToTime](#), [API_ConvertTimeToPosixTime](#)

Note

The block size and number of blocks will use the HPSS 64-bit type.

9.1.6.5 API_ConvertAttrsToVAttrs()

```
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	<i>Attrs</i>	Ptr to HPSS attributes
out	<i>VAttrs</i>	Ptr to VFS attributes

Return values

<i>None.</i>	

9.1.6.6 API_ConvertModeToPosixMode()

```
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	<i>Attrs</i>	HPSS object attributes
out	<i>PosixMode</i>	Pointer to posix mode_t

See also

[API_CalcBlock32](#), [API_CalcBlock64](#), [API_ConvertPosixModeToMode](#), [API_ConvertPosixTimeToTime](#),
[API_ConvertTimeToPosixTime](#)

9.1.6.7 API_ConvertPosixModeToMode()

```
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	<i>PosixMode</i>	POSIX mode_t to be translated
out	<i>Attrs</i>	Pointer to HPSS attributes

See also

[API_CalcBlock32](#), [API_CalcBlock64](#), [API_ConvertModeToPosixMode](#), [API_ConvertPosixTimeToTime](#),
[API_ConvertTimeToPosixTime](#)

9.1.6.8 API_ConvertPosixTimeToTime()

```
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	<i>Attrs</i>	HPSS object attributes
in	<i>Atime</i>	Last Access time
in	<i>Mtime</i>	Last Modify time
in	<i>Ctime</i>	Last Change time

See also

[API_CalcBlock32](#), [API_CalcBlock64](#), [API_ConvertModeToPosixMode](#) [API_ConvertPosixModeToMode](#),
[API_ConvertTimeToPosixTime](#)

9.1.6.9 API_ConvertTimeToPosixTime()

```
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	<i>Attrs</i>	HPSS object attributes
out	<i>Atime</i>	Last Access time
out	<i>Mtime</i>	Last Modify time
out	<i>Ctime</i>	Last Change time

See also

[API_CalcBlock32](#), [API_CalcBlock64](#), [API_ConvertModeToPosixMode](#) [API_ConvertPosixModeToMode](#),
[API_ConvertPosixTimeToTime](#)

9.1.6.10 API_FreeIOR()

```
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	<i>IORPtr</i>	pointer to IOR to free

9.1.6.11 API_GetBatchStatus()

```
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	<i>Status</i>	Batch Status Structure
in, out	<i>Rc</i>	Batch Status Indicator

Return values

<i>-EFAULT</i>	NULL Status Structure
<i>-EFAULT</i>	NULL Return Code
<i>-EINVAL</i>	Invalid Status List Length
<i>-EINVAL</i>	NULL Status List Structure
<i>0</i>	Success

Note

The status of the first failure is returned, but there could be other failures subsequent to that one.

9.1.6.12 API_GetClientAddr()

```
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	<i>SrcSinkPtr</i>	initial src/sink descriptor
in	<i>SrcSinkReplyPtr</i>	initial src/sink reply
in	<i>Offset</i>	offset within transfer
out	<i>RetAddr</i>	returned client length
out	<i>RetLength</i>	returned length
out	<i>RetDescPtr</i>	returned descriptor ptr
out	<i>RetReplyPtr</i>	returned reply ptr

Return values

0	Success
-EFAULT	Could not find client address

9.1.6.13 API_GetDataThreadStackSize()

```
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](#)

9.1.6.14 API_GetServerIDForSubsystem()

```
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	<i>SubsystemID</i>	subsystem id
in	<i>RequestID</i>	request id
out	<i>RetID</i>	returned ID

Return values

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

9.1.6.15 API_GetUniqueRequestID()

```
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.

9.1.6.16 API_RemoveRegisterValues()

```
uint64_t API_RemoveRegisterValues (
    uint64_t InitialValue,
    ... )
```

Remove values from a bit register.

Parameters

in	<i>InitialValue</i>	Initial Value
in	...	A series of bit positions to turn off within the provided <i>InitialValue</i> . 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](#)

9.1.6.17 API_StageBatchFree()

```
void API_StageBatchFree (
    hpss_stage_batch_t * Batch )
```

Free a Batch Stage Request Structure.

Free a Batch Stage Request Structure

Parameters

in, out	<i>Batch</i>	Batch Request Structure

9.1.6.18 API_StageBatchGetEntry()

```
int API_StageBatchGetEntry (
    hpss_stage_batch_t * Batch,
    int Idx,
    hpss_stage_t * Entry )
```

Retrieve Stage Batch Entry.

Parameters

in	<i>Batch</i>	Batch Request Structure
in	<i>Idx</i>	Batch List Index
out	<i>Entry</i>	Stage Batch Entry Structure

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EFAULT</i>	NULL Batch Structure Provided
<i>HPSS_EFAULT</i>	NULL Entry Pointer Provided
<i>HPSS_EINVAL</i>	Invalid List Index Provided
<i>HPSS_EFAULT</i>	No Batch List allocated

9.1.6.19 API_StageBatchInit()

```
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	<i>Batch</i>	Batch Request Structure
in	<i>len</i>	Batch Request Length

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EFAULT</i>	NULL Batch Structure Provided
<i>HPSS_ERANGE</i>	The size of the batch is larger than the legal size. Split the batch into multiple batches of <i>HPSS_MAX_STAGE_BATCH_LEN</i> or smaller and try again.
<i>HPSS_ENOMEM</i>	Could not allocate batch structure

Note

The list structure is initialized with all zero values and the correct list length

9.1.6.20 **API_StageBatchInsertBFHandle()**

```
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	<i>Batch</i>	Batch Request Structure
in	<i>Idx</i>	Batch List Index
in	<i>ObjHandle</i>	Bitfile Handle
in	<i>FromStorageLevel</i>	Level to Stage Data from
in	<i>ToStorageLevel</i>	Level to Stage Data to
in	<i>Offset</i>	Data Offset to Stage
in	<i>Length</i>	Length of Data to Stage
in	<i>Flags</i>	Stage Flags

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EFAULT</i>	NULL Batch Structure Provided
<i>HPSS_EFAULT</i>	NULL Object Handle Provided
<i>HPSS_EFAULT</i>	No Batch List allocated
<i>HPSS_EINVAL</i>	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)
- BFS_NO_CB (Do not report result to the provided callback)

9.1.6.21 API_StageBatchInsertBFObj()

```
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	<i>Batch</i>	Batch Request Structure
in	<i>Idx</i>	Batch List Index
in	<i>BFObj</i>	Bitfile Obj
in	<i>FromStorageLevel</i>	Level to Stage Data from
in	<i>ToStorageLevel</i>	Level to Stage Data to
in	<i>Offset</i>	Data Offset to Stage
in	<i>Length</i>	Length of Data to Stage
in	<i>Flags</i>	Stage Flags

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EFAULT</i>	NULL Batch Structure Provided
<i>HPSS_EFAULT</i>	NULL Bitfile ID Provided
<i>HPSS_EFAULT</i>	No Batch List allocated
<i>HPSS_EINVAL</i>	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)

- BFS_NO_CB (Do not report result to the provided callback)

9.1.6.22 API_StageBatchInsertFd()

```
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	<i>Batch</i>	Batch Request Structure
in	<i>Idx</i>	Batch List Index
in	<i>Fd</i>	Open File Descriptor
in	<i>FromStorageLevel</i>	Level to Stage Data from
in	<i>ToStorageLevel</i>	Level to Stage Data to
in	<i>Offset</i>	Data Offset to Stage
in	<i>Length</i>	Length of Data to Stage
in	<i>Flags</i>	Stage Flags

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EFAULT</i>	NULL Batch Structure Provided
<i>HPSS_EFAULT</i>	No Batch List allocated
<i>HPSS_EINVAL</i>	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)
- BFS_NO_CB (Do not report result to the provided callback)

9.1.6.23 API_StageStatusFree()

```
void API_StageStatusFree (
    hpss_stage_batch_status_t * Status )
```

Free a Batch Stage Status Structure.

Free a Batch Stage Status Structure

Parameters

in, out	<i>Status</i>	Batch Status Structure

9.1.6.24 hpss_AbortIOByMoverId()

```
int hpss_AbortIOByMoverId (
    int32_t SubsystemId,
    hpss_svr_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	<i>SubsystemId</i>	Subsystem Id
in	<i>MoverId</i>	Mover Server Id

Return values

0	Success
<i>Other</i>	Error Code indicating the nature of the problem

Note

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

9.1.6.25 hpss_AbortIOByMoverName()

```
int hpss_AbortIOByMoverName (
    int32_t SubsystemId,
    char * MoverName )
```

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

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

Parameters

in	<i>SubsystemId</i>	Subsystem Id
in	<i>MoverName</i>	Mover Descriptive Name

Return values

0	Success
<i>Other</i>	Error Code indicating the nature of the problem

Note

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

9.1.6.26 hpss_AbortIOByRequestId()

```
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	<i>SubsystemId</i>	Subsystem ID
in	<i>RequestIdToAbort</i>	Request ID
in	<i>Ucred</i>	User Credentials (optional)

Return values

0	Success
<i>Other</i>	Error Code indicating the nature of the problem

9.1.6.27 hpss_AbortIOByRequestMatch()

```
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	<i>SubsystemId</i>	Subsystem ID
in	<i>PartialReqString</i>	Partial request ID to abort
in	<i>Ucred</i>	User Credentials (optional)

Return values

0	Success
<i>HPSS_EFAULT</i>	No request string was provided
<i>HPSS_EINVAL</i>	More than one match
<i>HPSS_RANGE</i>	Partial request string was too long
<i>Other</i>	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 inadvertently 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.

9.1.6.28 hpss_AbortIOByUserId()

```
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	<i>SubsystemId</i>	Subsystem ID
in	<i>Ucred</i>	User Credentials
out	<i>RequestsAborted</i>	Requests Aborted
out	<i>RequestsAlreadyAborted</i>	Requests Already Aborted
out	<i>RequestsFound</i>	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: $\text{RequestsAborted} - \text{RequestsAlreadyAborted}$.

9.1.6.29 hpss_Access()

```
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	<i>Path</i>	Path of file to check access rights
in	<i>Amode</i>	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 <i>Amode</i> are denied, or search permission is denied on a component path of the path prefix.
-EFAULT	The <i>Path</i> parameter is a NULL pointer.
-EINVAL	An invalid value was specified for <i>Amode</i> .
-ENAMETOOLONG	The length of the <i>Path</i> 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 <i>Path</i> argument points to an empty string.
-ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

See also

[hpss_AccessHandle](#), [hpss_Chmod](#), [hpss_FileSetAttributes](#)

9.1.6.30 hpss_AccessHandle()

```
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_AccessHandle` 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	<i>ObjHandle</i>	Handle to the parent object
in	<i>Path</i>	Path name of the file being checked
in	<i>Amode</i>	Indicates the type of file access being checked. Refer to POSIX.1 for possible values.
in	<i>Ucred</i>	User credentials

Return values

0	No error, caller has access.
-EACCES	The permissions specified by <i>Amode</i> are denied, or search permission is denied on a component of the path prefix.
-EINVAL	An invalid value was specified for <i>Amode</i> or the <i>Path</i> parameter is a NULL pointer.
-ENAMETOOLONG	The length of the <i>Path</i> 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 <i>Path</i> argument points to an empty string.
-ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

See also

[hpss_Access](#), [hpss_Chmod](#), [hpss_FileSetAttributes](#)

9.1.6.31 hpss_AcctCodeToName()

```
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	<i>AcctCode</i>	Account Code to look up
in, out	<i>Site</i>	UUID of the Site; if NULL the default site for the current working directory is used
out	<i>AcctName</i>	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](#)

9.1.6.32 hpss_AcctNameToCode()

```
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	<i>AcctName</i>	Account Name to look up
in, out	<i>Site</i>	UUID of the site; if NULL then the default site is used.
out	<i>AcctCode</i>	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](#)

9.1.6.33 hpss_BuildLevelString()

```
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

<i>ENOMEM</i>	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.

9.1.6.34 hpss_Chacct()

```
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	<i>Path</i>	Name of the file whose account code will be modified
in	<i>AcctCode</i>	New accounting code for the file at <i>Path</i>

Return values

<i>-EACCES</i>	Search permission is denied on a component of the <i>Path</i> prefix.
<i>-EFAULT</i>	<i>Path</i> is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the <i>Path</i> argument points to an empty string.
<i>-ENOTDIR</i>	A component of the <i>Path</i> prefix is not a directory.

Return values

<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation or is configured for Unix-style accounting.
<i>0</i>	No error. Account code altered as directed.

See also

[hpss_ChacctByName](#)

9.1.6.35 **hpss_ChacctByName()**

```
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	<i>Path</i>	File or directory for which the account code is being changed
in	<i>AcctName</i>	Account name corresponding to the new account code for the file or directory

Return values

<i>0</i>	No error. Account name altered as directed.
<i>-EACCES</i>	Search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	Path is a NULL pointer or AcctName is NULL.
<i>-EINVAL</i>	The specified AccountName is not a valid account name.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation or is configured for Unix-style accounting.

9.1.6.36 **hpss_Chdir()**

```
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	<i>Path</i>	Path to the directory

Return values

0	No error. New working directory set properly.
-EACCES	Search permission is denied on a component of the <i>Path</i> prefix.
-EFAULT	The <i>Path</i> parameter is a NULL pointer.
-ENAMETOOLONG	The length of the <i>Path</i> 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 <i>Path</i> argument points to an empty string.
-ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

9.1.6.37 hpss_Chmod()

```
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	<i>Path</i>	path to the object
in	<i>Mode</i>	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.

9.1.6.38 hpss_Chown()

```
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	<i>Path</i>	path to the object
in	<i>Owner</i>	desired new owner ID
in	<i>Group</i>	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.

9.1.6.39 hpss_Chroot()

```
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	<i>Path</i>	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.

Return values

<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-ENOMEM</i>	Memory allocation failure

Note

As currently implemented, symbolic links allow a client to access files outside the new root directory.

9.1.6.40 hpss_ClearLastHPSSErrno()

```
void hpss_ClearLastHPSSErrno (
    void )
```

Clears the current HPSS errno state.

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

9.1.6.41 hpss_ClientAPIInit()

```
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

<i>0</i>	Success
<i>-ENOCNNECT</i>	Could not initialize HPSS connection services.
<i>-ENOMEM</i>	Memory allocation failed.
<i>-EPERM</i>	Could not initialize security.

9.1.6.42 hpss_ClientAPIReset()

```
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](#)

9.1.6.43 hpss_Close()

```
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	<i>Fildes</i>	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](#)

9.1.6.44 hpss_Closedir()

```
int hpss_Closedir (
    int Dirdes )
```

Close an open directory stream.

Parameters

in	<i>Dirdes</i>	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.

Return values

<i>-EBADF</i>	The specified directory descriptor does not refer to an open directory.
<i>-EBUSY</i>	The directory is currently in use by another client thread.

See also

hpss_OpenDir

9.1.6.45 hpss_ConvertIdsToNames()

```
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	<i>NumEntries</i>	# entries in Specs
in, out	<i>Specs</i>	Principals to convert

Return values

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

9.1.6.46 hpss_ConvertNamesToIds()

```
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	<i>NumEntries</i>	Number of entries in Specs
in, out	<i>Specs</i>	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.
-ENOCNNECT	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.

9.1.6.47 hpss_CopyFile()

```
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	<i>SrcFildes</i>	open desc for src file
in	<i>DestFildes</i>	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.

9.1.6.48 hpss_CopyrightStrings()

```
char** hpss_CopyrightStrings (
    void )
```

Read the copyright file and return the strings.

Returns

An array of strings read from the copyright file

9.1.6.49 hpss_Creat()

```
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 *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	<i>Path</i>	Path to file to be opened
in	<i>Mode</i>	Desired file perms if create
in	<i>HintsIn</i>	Desired class of service
in	<i>HintsPri</i>	Priorities of hint struct
out	<i>HintsOut</i>	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.

9.1.6.50 hpss_Create()

```
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 'Path', with permissions as specified by 'Mode' and using the class of service values specified by 'HintsIn' and 'HintsPri', if non-NULL.

Parameters

in	<i>Path</i>	Path to file to be opened
in	<i>Mode</i>	Desired file perms if create
in	<i>HintsIn</i>	Desired class of service
in	<i>HintsPri</i>	Priorities of hint struct
out	<i>HintsOut</i>	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.

9.1.6.51 hpss_CreateHandle()

```
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 *Path*, with permissions as specified by *Mode* and using the class of service values specified by *HintsIn* and *HintsPri*, if non-NULL.

Parameters

in	<i>ObjHandle</i>	Parent object handle
in	<i>Path</i>	Path to file to be opened
in	<i>Mode</i>	Desired file perms if create
in, out	<i>Ucred</i>	User credentials
in	<i>HintsIn</i>	Desired class of service
in	<i>HintsPri</i>	Priorities of hint struct
out	<i>HintsOut</i>	Granted class of service
out	<i>AttrsOut</i>	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.

9.1.6.52 hpss_DeleteACL()

```
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	<i>Path</i>	Names the file for which the <i>ACL</i> is being removed.
in	<i>Options</i>	Bit vector used to specify what type of ACL is to be removed. One of: • - <i>HPSS_ACL_OPTION_OBJ</i> Normal ACL • - <i>HPSS_ACL_OPTION_IO</i> Initial Object ACL (only valid for directory objects) • - <i>HPSS_ACL_OPTION_IC</i> Initial Container ACL (only valid for directory objects)
in	<i>ACL</i>	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.

Return values

<i>-EINVAL</i>	Exactly one of the <code>HPSS_ACL_OPTION_*</code> bits must be set in the Options bit vector.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.

See also

[hpss_GetACL](#) [hpss_UpdateACL](#) [hpss_SetACL](#)

9.1.6.53 hpss_DeleteACLHandle()

```
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	<i>ObjHandle</i>	Parent object handle
in	<i>Path</i>	File for which the ACL is being removed
in	<i>Ucred</i>	User credentials
in	<i>Options</i>	Bit vector used to specify what type of ACL is to be removed. One of: • - <code>HPSS_ACL_OPTION_OBJ</code> Normal ACL • - <code>HPSS_ACL_OPTION_IO</code> Initial Object ACL (only valid for directory objects) • - <code>HPSS_ACL_OPTION_IC</code> Initial Container ACL (only valid for directory objects)
in	<i>ACL</i>	Points to the list of ACL entries to be removed.

Return values

0	No error.
<i>-EACCES</i>	Search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path or ACL parameter is a NULL pointer.
<i>-EINVAL</i>	Exactly one of the <code>HPSS_ACL_OPTION_*</code> bits must be set in the Options bit vector.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

Return values

<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.
<i>-ESRCH</i>	A specified ACL entry did not match an existing ACL entry for the file.

See also

[hpss_GetACLHandle](#) [hpss_UpdateACLHandle](#) [hpss_SetACLHandle](#)

9.1.6.54 **hpss_Fclear()**

```
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	<i>Fildes</i>	Specifies the file descriptor identifying the open file
in	<i>Length</i>	Number of bytes to be cleared

Return values

0	Success
<i>-EBADF</i>	The specified file descriptor does not correspond to a file opened for writing.
<i>-EBUSY</i>	The specified file descriptor is currently in use.

See also

[hpss_FclearOffset](#) [hpss_Ftruncate](#)

9.1.6.55 **hpss_FclearOffset()**

```
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	<i>Fildes</i>	Specifies the file descriptor identifying the open file
in	<i>Offset</i>	Specifies where to begin clearing the file
in	<i>Length</i>	Specifies the number of bytes to be cleared

Return values

0	Success
-EINVAL	The <i>Offset</i> or <i>Length</i> 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](#)

9.1.6.56 hpss_Fclose()

```
int hpss_Fclose (
    HPSS_FILE * hpss_Stream )
```

Closes a stream and cleans up.

Parameters

in, out	<i>hpss_Stream</i>	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	Permission denied.
-EBADF	The file descriptor underlying the Stream is not valid.
-EBUSY	Some other thread is manipulating this file.
-EINVAL	Invalid (NULL) <i>hpss_Stream</i> parameter.
-EIO	An internal error occurred.

Note

Resources Used: Memory for I/O buffers and HPSS_FILE structure.

See also

[hpss_Fflush](#)

9.1.6.57 hpss_Fcntl()

```
int hpss_Fcntl (
    int hpss_fd,
    int Cmd,
    long Arg )
```

Set or get file control arguments.

Parameters

in	<i>hpss↔ _fd</i>	HPSS file descriptor
in	<i>Cmd</i>	Requested command
in	<i>Arg</i>	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

Return values

<i>-EINVAL</i>	Invalid argument value
<i>-EIO</i>	Internal I/O error
<i>-EFAULT</i>	Invalid object handle
<i>-ENOTSUP</i>	Command not supported
<i>-EBADF</i>	Bad file descriptor
<i>-ENOMEM</i>	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.

9.1.6.58 hpss_FdelFileDigestList()

```
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	<i>Fildes</i>	Open bitfile descriptor
in	<i>Flags</i>	Digest deletion preference

Return values

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

9.1.6.59 hpss_Fflush()

```
int hpss_Fflush (
    HPSS_FILE * hpss_Stream )
```

Writes buffered data for a stream.

Parameters

<i>in, out</i>	<i>hpss_Stream</i>	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

<i>0</i>	No error. The buffer was flushed.
<i>-EACCES</i>	Permission denied.
<i>-EBADF</i>	The file descriptor underlying the Stream is not valid.
<i>-EINVAL</i>	Invalid (NULL) <i>hpss_Stream</i> parameter.
<i>-ESTALE</i>	Stale connections
<i>-EIO</i>	An internal error occurred.

See also

[hpss_Fopen](#) [hpss_Fclose](#)

9.1.6.60 hpss_Fgetc()

```
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

<i>in, out</i>	<i>stream</i>	stream from which to receive data

Return values

<i>Positive</i>	Character read from the data stream
<i>EOF</i>	End of file reached.
<i>-EBADF</i>	Invalid file descriptor or file not open.
<i>-EINVAL</i>	Invalid input parameter (Ptr Null or Size zero)
<i>-EFAULT</i>	NULL stream pointer.

Return values

-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.

9.1.6.61 hpss_FgetFileDigest()

```
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	<i>Fildes</i>	Open bitfile descriptor
in, out	<i>*Digest</i>	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](#)

9.1.6.62 hpss_FgetFileDigestList()

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

Return values

0	Success
-EBADF	<i>Fildes</i> is not a valid open descriptor
-ENOENT	Digest information doesn't exist
-EBUSY	<i>Fildes</i> is a busy descriptor
-ESTALE	<i>Fildes</i> is a stale descriptor and should be reopened
-EFAULT	<i>DigestList</i> 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_HASH_DIGEST_ACTIVE flag, which is set or cleared by the client. For single stripe checksums, this flag is the same as the HPSS_FILE_HASH_DIGEST_VALID flag since the system manages the hash validity state.

See also

hpss_GetFileDigestHandle, [hpss_FgetFileDigest](#)

9.1.6.63 hpss_Fgets()

```
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	<i>s</i>	Buffer in which to receive data
in	<i>n</i>	number of bytes to read
in	<i>stream</i>	stream from which to receive data

Return values

<i>Not</i>	Null String Data In Buffer
<i>Null</i>	String End of File or Error Reached
<i>EBUSY</i>	Errno may be set to EBUSY if the HPSS file table entry is busy.
<i>ESTALE</i>	Errno may be set to ESTALE if the connection to HPSS has become stale.
<i>EPERM</i>	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.
<i>EIO</i>	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.

9.1.6.64 hpss_FileGetAttributes()

```
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 referred to by *Path*.

Parameters

in	<i>Path</i>	Path to the object
out	<i>AttrOut</i>	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.

9.1.6.65 hpss_FileGetAttributesBitfile()

```
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 referred to by 'Bitfile'. It returns one of possibly multiple path names to the bitfile and the file attributes structure for the bitfile.

Parameters

in	<i>BitfileObj</i>	Bitfile Object
out	<i>Path</i>	Path to the object
out	<i>AttrOut</i>	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.

Return values

<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	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.

9.1.6.66 **hpss_FileGetAttributesHandle()**

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path to the object
in	<i>Ucred</i>	user credentials
out	<i>AttrOut</i>	attributes after query

Return values

0	No error, caller has access.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The Path or AttrOut parameter is a NULL pointer.
<i>-EINVAL</i>	ObjHandle is NULL.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	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.

9.1.6.67 hpss_FileGetDigestHandle()

```
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	* <i>ObjHandle</i>	parent object handle
in	* <i>Path</i>	path of file to get attributes
in	* <i>Ucred</i>	user credentials
in, out	* <i>Digest</i>	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 indicate 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](#)

9.1.6.68 hpss_FileGetDigestListHandle()

```
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	* <i>ObjHandle</i>	parent object handle
in	* <i>Path</i>	path of file to get attributes
in	* <i>Ucred</i>	user credentials
in	<i>Flags</i>	Stripe preferencing
out	* <i>StripeLength</i>	Length of each stripe, in bytes • If the digest is not striped this value is unused.
out	* <i>DigestList</i>	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](#)

9.1.6.69 hpss_FileGetXAttributes()

```
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 referred 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	<i>Path</i>	Path to the object
in	<i>Flags</i>	Flags for storage attrs • API_GET_STATS_FOR_LEVEL - Returns bitfile attributes at the storage level specified by the <i>StorageLevel</i> 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	<i>StorageLevel</i>	Storage level to query
out	<i>AttrOut</i>	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.SCAattrib[i].NumberOfVVs; j++)
```

```

{
    if (AttrOut.SCAattrib[i].VVAttrib[j].PVList != NULL)
    {
        free(AttrOut.SCAattrib[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.

9.1.6.70 hpss_FileGetXAttributesHandle()

```

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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path to the object
in	<i>Ucred</i>	user credentials
in	<i>Flags</i>	flags for storage attrs
in	<i>StorageLevel</i>	level to query
out	<i>AttrOut</i>	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.

9.1.6.71 hpss_FileSetAttributes()

```
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 referred to by 'Path'.

Parameters

in	<i>Path</i>	path to the object

Parameters

in	<i>SetFlags</i>	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
		Generated on Tue Sep 23 2025 for HPSS 11.3 Programmer's Reference • CORE_ATTR_USER_PERMS • CORE_ATTR_WRITE_COUNT

Parameters

in	<i>AttrIn</i>	input attributes
out	<i>AttrOut</i>	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.

9.1.6.72 `hpss_FileSetAttributesBitfile()`

```
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 referred to by 'BitfileID'. It returns one of possibly multiple path names to the bitfile and the file attributes structure for the entry.

Parameters

in	<i>BitfileObj</i>	Bitfile Object
in	<i>SelFlags</i>	attributes fields to set; refer to <code>hpss_SetFileAttributes</code> for a list
in	<i>AttrIn</i>	input attributes
out	<i>AttrOut</i>	attributes after query
out	<i>Path</i>	path to the object

Return values

<i>0</i>	No error, caller has access.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The AttrOut parameter is a NULL pointer.
<i>-EINVAL</i>	An attribute value or selection flag is invalid, or BitfileObj is NULL.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOSPC</i>	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.

9.1.6.73 `hpss_FileSetAttributesHandle()`

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path to the object
in	<i>Ucred</i>	user credentials
in	<i>SelFlags</i>	attributes fields to set; refer to <code>hpss_SetFileAttributes</code> for a list
in	<i>AttrIn</i>	input attributes
out	<i>AttrOut</i>	attributes after change

Return values

<i>0</i>	No error, caller has access.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.

Return values

-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.

9.1.6.74 `hpss_FileSetCOS()`

```
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 referred 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	<i>Path</i>	path to the object
in	<i>COSId</i>	change COS to this value
in	<i>StreamId</i>	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.

Return values

<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOSPC</i>	Resources could not be allocated to satisfy the request.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EOPNOTSUPP</i>	The requested change is not supported.
<i>-EPERM</i>	The client does not have the appropriate privileges to change the file's attributes.

9.1.6.75 hpss_FileSetCOSHandle()

```
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 referred 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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path to the object
in	<i>Ucred</i>	user credentials
in	<i>COSId</i>	change COS to this value
in	<i>StreamId</i>	used this Core Server stream

Return values

0	No error, caller has access.
<i>-ENOMEM</i>	Not Enough Memory
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	An attribute value or selection flag is invalid.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOSPC</i>	Resources could not be allocated to satisfy the request.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EOPNOTSUPP</i>	The requested change is not supported.
<i>-EPERM</i>	The client does not have the appropriate privileges to change the file's attributes.

9.1.6.76 hpss_FilesetCreate()

```
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	<i>CoreServerID</i>	Core Server ID
in	<i>CreateOptions</i>	creation options CORE_NS_FILESET - Create Core Server fileset metadata
in	<i>FilesetAttrBits</i>	Fileset attributes to set
in	<i>FilesetAttrs</i>	Fileset attributes
in	<i>ObjectAttrBits</i>	Object attributes to set
in	<i>ObjectAttrs</i>	Object attributes
out	<i>RetFilesetAttrBits</i>	Fileset attr to return
out	<i>RetObjectAttrBits</i>	Dir attr to return
out	<i>RetFilesetAttrs</i>	Returned fileset attr
out	<i>RetObjectAttrs</i>	Returned dir attr
out	<i>FilesetHandle</i>	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.

9.1.6.77 hpss_FilesetDelete()

```
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	<i>FilesetName</i>	Fileset name
in	<i>FilesetId</i>	Fileset ID
in	<i>FilesetHandle</i>	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 than 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.

9.1.6.78 hpss_FilesetGetAttributes()

```
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	<i>FilesetName</i>	Fileset name
in	<i>FilesetId</i>	Fileset ID
in	<i>FilesetHandle</i>	Fileset object handle
in	<i>CoreServerID</i>	Core server ID for fileset
in	<i>FilesetAttrBits</i>	Fileset attribute bits
out	<i>FilesetAttrs</i>	Returned attributes

Return values

<i>-EACCES</i>	The user is not the root user or a trusted user.
<i>-EFAULT</i>	The FilesetName, FilesetId and FilesetHandle arguments are all NULL pointers.
<i>-EINVAL</i>	More than one type of fileset identifier was specified or invalid file set attribute bits were specified.
<i>-ENOENT</i>	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.

9.1.6.79 hpss_FilesetListAll()

```
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	<i>OffsetIn</i>	offset to start at
in	<i>Entries</i>	number of entries to return
out	<i>End</i>	hit end of the list
out	<i>OffsetOut</i>	offset of last entry
out	<i>FSentPtr</i>	fileset entry information

Return values

0	Success
<i>-EFAULT</i>	The End, OffsetOut or FSentPtr parameter is a NULL pointer.
<i>-EINVAL</i>	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 allocated space. 'OffsetOut' is the point that the lookup is at when it accumulated the specified number of entries. 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 all the entries were accumulated.

9.1.6.80 hpss_FilesetSetAttributes()

```
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	<i>FilesetName</i>	Fileset name
in	<i>FilesetId</i>	Fileset ID
in	<i>FilesetHandle</i>	Fileset object handle
in	<i>FilesetAttrBitsIn</i>	Fileset attribute bits
in	<i>FilesetAttrsIn</i>	Fileset attributes
in	<i>FilesetAttrBitsOut</i>	Returned set attribute bits
out	<i>FilesetAttrsOut</i>	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 than 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.

9.1.6.81 hpss_Fopen()

```
HPSS_FILE* hpss_Fopen (
    const char * Path,
    const char * Mode )
```

Opens an HPSS file and associates a stream with it.

Parameters

in	<i>Path</i>	Path/name of file
in	<i>Mode</i>	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

<i>EFAULT</i>	Path parameter is NULL
<i>EINVAL</i>	Invalid mode parameter
<i>EIO</i>	Internal error
<i>ENOENT</i>	Path parameter points to an empty string
<i>ENOMEM</i>	Memory allocation failure
<i>EMFILE</i>	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.

9.1.6.82 `hpss_Fpreallocate()`

```
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	<i>Fildes</i>	ID of open object
in	<i>Length</i>	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.

9.1.6.83 hpss_Fread()

```

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	<i>Ptr</i>	Pointer to input array.
in	<i>Size</i>	Size in bytes of data items
in	<i>Num</i>	Number of data items to read
in, out	<i>hpss_Stream</i>	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 occurred.
ENOENT	File doesn't exist for reading.
ENOMEM	Memory allocation failure.

9.1.6.84 hpss_Fseek()

```
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	<i>hpss_Stream</i>	Pointer to HPSS_FILE
in	<i>OffsetIn</i>	Bytes to add/subtract
in	<i>Whence</i>	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()

9.1.6.85 hpss_FsetFileDigest()

```
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	<i>Fildes</i>	Open bitfile descriptor
in	<i>Digest</i>	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.

9.1.6.86 hpss_FsetFileDigestList()

```
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 Integrity (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	<i>Fildes</i>	Open bitfile descriptor
in	<i>StripeLength</i>	Stripe Length for digests
in	<i>DigestList</i>	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.

9.1.6.87 hpss_Fstat()

```
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	<i>Fildes</i>	ID of open file/directory
out	<i>Buf</i>	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_OpenBitfileVAttr 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.

9.1.6.88 hpss_Fsync()

```
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	<i>hpss_fd</i>	valid HPSS file descriptor

Return values

0	hpss_fd is a valid file descriptor.
-EBADF	hpss_fd is not a valid file descriptor.

9.1.6.89 hpss_Ftell()

```
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	<i>hpss_Stream</i>	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

9.1.6.90 hpss_Ftruncate()

```
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	<i>Fildes</i>	ID of open object
in	<i>Length</i>	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.

9.1.6.91 hpss_Fwrite()

```
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	<i>Ptr</i>	Pointer to input array.
in	<i>Size</i>	Size in bytes of data items
in	<i>Num</i>	Number of data items to write
in, out	<i>hpss_Stream</i>	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

<code>0</code>	Successful write operation.
<code>-EACCES</code>	Permission denied.
<code>-EBADF</code>	The file descriptor underlying the stream is not valid.
<code>-EBUSY</code>	Another thread is accessing this table entry.
<code>-EFBIG</code>	Size of write request > 2GB
<code>-EINVAL</code>	NULL <code>hpss_Stream</code> parameter.
<code>-EIO</code>	An internal HPSS error occurred.
<code>-ENOENT</code>	File doesn't exist for writing.
<code>-ENOMEM</code>	Memory allocation failure.
<code>-ESTALE</code>	Connection not valid for this file table entry.

9.1.6.92 `hpss_GetAcct()`

```
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	<i>RetDefAcct</i>	default account code
out	<i>RetCurAcct</i>	current account code

Return values

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

9.1.6.93 hpss_GetAcctName()

```
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	<i>AcctName</i>	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.

9.1.6.94 hpss_GetACL()

```
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	<i>Path</i>	path of file
in	<i>Options</i>	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	<i>ACL</i>	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.

9.1.6.95 hpss_GetACLHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file
in	<i>Ucred</i>	user credentials
in	<i>Options</i>	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	<i>ACL</i>	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.

Return values

<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	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.

9.1.6.96 hpss_GetAllSubsystems()

```
int hpss_GetAllSubsystems (
    ls_map_array_t ** ls_map_array )
```

Retrieve a map of all subsystems.

Parameters

out	<i>ls_map_array</i>	Location Map Array

This function finds all subsystems for the current site.

Return values

<i>0</i>	Success
<i>HPSS_ENOENT</i>	No subsystems were returned.
<i>Other</i>	Error communicating with the LS

Note

The caller is responsible for freeing the returned map.

9.1.6.97 hpss_GetAsyncStatus()

```
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	<i>Call↔ BackId</i>	callback request identifier
in	<i>BitfileObj</i>	Bitfile Object
out	<i>Status</i>	status HPSS_STAGE_STATUS_UNKNOWN - No status available HPSS_STAGE_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.

9.1.6.98 hpss_GetAttrHandle()

```
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	<i>ObjHandle</i>	Parent object handle
in	<i>Path</i>	Path of file to get attributes
in	<i>Ucred</i>	User credentials
out	<i>HandleOut</i>	Returned object handle for <i>Path</i>
out	<i>AttrOut</i>	Returned attributes for <i>Path</i>

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.

9.1.6.99 hpss_GetBatchAsynchStatus()

```
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	<i>CallbackId</i>	Callback Id
in	<i>BFObjs</i>	Bitfile Objects to Status
in	<i>Type</i>	This is no longer used / the position in the queue is always returned
out	<i>Status</i>	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.

Deprecated This function will be removed in a future version of HPSS. Use [hpss_GetCallbackStatus\(\)](#).

9.1.6.100 hpss_GetCallbackStatus()

```
int hpss_GetCallbackStatus (
    int SubsystemId,
    hpss_reqid_t CallbackId,
    hpss_cb_status_t * Status )
```

Check the status of files in a stage callback request.

Provide the status (Not Found, Queued, or Active) for each bitfile using the callback id.

Parameters

in	<i>SubsystemId</i>	Subsystem Id (0 for default subsystem)
in	<i>CallbackId</i>	Callback Id
out	<i>Status</i>	Batch Status

Return values

0	Status successfully retrieved
-EFAULT	No Status structure provided
Other	Status could not be retrieved

9.1.6.101 hpss_GetConfiguration()

```
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	<i>ConfigOut</i>	ptr for returned attributes

Return values

0	Success
-EFAULT	The ConfigOut parameter is NULL.

9.1.6.102 hpss_Getcwd()

```
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	<i>Buf</i>	current working directory
in	<i>Size</i>	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.

9.1.6.103 hpss_Getdents()

```
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	<i>ObjHandle</i>	directory object handle
in	<i>OffsetIn</i>	directory position
in	<i>Ucred</i>	user credentials
in	<i>BufferSize</i>	size of output buffer
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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\(\)](#).

9.1.6.104 hpss_GetdentsAttrs()

```
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	<i>ObjHandle</i>	directory object handle
in	<i>OffsetIn</i>	directory position
in	<i>Ucred</i>	user credentials
in	<i>BufferSize</i>	size of output buffer
in	<i>GetAttributes</i>	get object attributes?
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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	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\(\)](#).

9.1.6.105 hpss_GetDistFile()

```
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	<i>Fildes</i>	Open file handle
out	<i>FileInfo</i>	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.

9.1.6.106 hpss_GetFileNotrunc()

```
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	<i>Path</i>	Pathname of file
out	<i>NotruncFlag</i>	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.

9.1.6.107 hpss_GetFullPath()

```
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	<i>ObjHandle</i>	Object Handle
out	<i>FullPath</i>	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*.

9.1.6.108 hpss_GetFullPathBitfile()

```
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	<i>BitfileObj</i>	Bitfile Object Handle
out	<i>FullPath</i>	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*.

9.1.6.109 hpss_GetFullPathBitfileLen()

```
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	<i>BitfileObj</i>	Bitfile Object Handle
in, out	<i>Path</i>	Path to object
in	<i>BufLen</i>	Size of <i>Path</i>

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.

9.1.6.110 hpss_GetFullPathHandle()

```
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	<i>ObjHandle</i>	Object Handle
in	<i>Path</i>	Additional Path
out	<i>FullPath</i>	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*.

9.1.6.111 hpss_GetFullPathHandleLen()

```
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	<i>ObjHandle</i>	Object Handle
in	<i>ObjPath</i>	Additional Path
in, out	<i>Path</i>	Path to object
in	<i>BufLen</i>	Size of <i>Path</i>

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.

9.1.6.112 hpss_GetFullPathLen()

```
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	<i>ObjHandle</i>	Object Handle
in, out	<i>Path</i>	Path to object
in	<i>BufLen</i>	Size of <i>Path</i>

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.

9.1.6.113 `hpss_GetGID()`

```
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() .

9.1.6.114 `hpss_GetIntersectedList()`

```
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	<i>List</i>	List of all items to process
in	<i>Partial</i>	Sublist of items to include

Returns

A new allocation of list items

Note

The new list array must be freed

9.1.6.115 hpss_GetJunctionAttrs()

```
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	<i>Path</i>	path of file to get statistics
out	<i>AttrOut</i>	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.

9.1.6.116 hpss_GetJunctions()

```
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	<i>SubsystemID</i>	Subsystem identifier
in	<i>OffsetIn</i>	offset to start at
in	<i>Entries</i>	number of entries to return
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	offset of last entry
out	<i>JentPtr</i>	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.

9.1.6.117 hpss_GetLastHPSSErrno()

```

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.

9.1.6.118 hpss_GetListAttrs()

```

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	<i>Path</i>	path of file to get statistics
out	<i>AttrOut</i>	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.

9.1.6.119 `hpss_GetNextIORequestID()`

```
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

<i>This</i>	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.

9.1.6.120 `hpss_GetObjFromHandle()`

```
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	<i>ObjHandle</i>	parent object handle
in, out	<i>ObjHash</i>	object id / hash pair

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>-EFAULT</i>	The <i>ObjHandle</i> parameter is a NULL pointer.

9.1.6.121 hpss_GetObjType()

```
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	<i>ObjHandle</i>	parent object handle

Return values

<i>-EFAULT</i>	The <i>ObjHandle</i> parameter is a NULL pointer.
<i>-EINVAL</i>	The <i>ObjHandle</i> points to an invalid object.
<i>other</i>	Object type

9.1.6.122 hpss_GetPathHandle()

```
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	<i>ObjHandle</i>	object handle
out	<i>FilesetRootHandle</i>	Fileset root handle
out	<i>Path</i>	Path to object

Return values

<i>-EFAULT</i>	The Path or FilesetRootHandle parameter is NULL.
<i>-EINVAL</i>	The ObjHandle is NULL.

9.1.6.123 hpss_GetPathHandleObjHash()

```
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	<i>ObjIdHash</i>	object hash
in	<i>SubsysId</i>	subsystem id
out	<i>FilesetRootHandle</i>	Fileset root handle
out	<i>ObjHandle</i>	object handle
out	<i>Path</i>	Path to object

Return values

0	Success
<i>-EFAULT</i>	Path pointer is NULL
<i>-ENAMETOOLONG</i>	The path name could not be retrieved because it is too long.
<i>-ENOENT</i>	The path could not be retrieved because no path

9.1.6.124 hpss_GetPathObjHash()

```
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	<i>ObjIdHash</i>	object id / hash pair
in	<i>SubsysId</i>	subsystem id
out	<i>FilesetRootHandle</i>	Fileset root handle
out	<i>Path</i>	Path to object

Return values

<i>-EFAULT</i>	Path pointer is NULL
<i>-ENAMETOOLONG</i>	The path name could not be retrieved because it is too long.
<i>-ENOENT</i>	The path could not be retrieved because no path corresponds to the given object id.

9.1.6.125 hpss_GetRawAttrHandle()

```
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	<i>ObjHandle</i>	Parent object handle
in	<i>Path</i>	Path of file to get attributes
in	<i>Ucred</i>	User credentials
out	<i>HandleOut</i>	Returned object handle for <i>Path</i>
out	<i>AttrOut</i>	Returned attributes for <i>Path</i>

Return values

<i>0</i>	No error. Valid information returned.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The Path or AttrOut parameter is a NULL pointer.
<i>-EINVAL</i>	The ObjHandle parameter is NULL.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.

9.1.6.126 hpss_GetSubSysLimits()

```
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	<i>SubsystemID</i>	Identifier for the subsystem
in	<i>Ucred</i>	User Credentials
out	<i>LimitsOut</i>	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.

9.1.6.127 hpss_GetSubSysStats()

```
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	<i>SubsystemID</i>	identifier for the subsystem
out	<i>StatsOut</i>	subsystem statistics information

Return values

0	No error.
-EFAULT	The StatsOut parameter is a NULL pointer.

9.1.6.128 hpss_GetThreadUcred()

```
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	<i>RetUcred</i>	credentials on client

Return values

0	No error. The RetUcred contains the current credentials for the user.
-EFAULT	RetUcred is a NULL pointer.

9.1.6.129 hpss_GetTrashcan()

```
int hpss_GetTrashcan (
    const char * Path,
    char * TrashcanBuf,
    int TrashcanBufLength,
    ns_ObjHandle_t * FilesetRoot )
```

Locate the trashcan for a provided path.

Parameters

in	<i>Path</i>	Path to lookup trashcan for
in, out	<i>TrashcanBuf</i>	Buffer for trashcan path
in	<i>TrashcanBufLength</i>	TrashcanBuf's length
in, out	<i>FilesetRoot</i>	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

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EFAULT</i>	Path or Trashcan Buffer not provided.
<i>HPSS_ENOENT</i>	Path does not exist
<i>HPSS_ENOTDIR</i>	Some component of the path is not a directory
<i>HPSS_ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit
<i>HPSS_EINVAL</i>	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.

9.1.6.130 hpss_GetTrashcanHandle()

```
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	<i>ObjHandle</i>	Object Handle for lookup
in	<i>Path</i>	Path to lookup trashcan for
in	<i>Ucred</i>	User Credentials
in, out	<i>TrashcanBuf</i>	Buffer for trashcan path
in	<i>TrashcanBufLength</i>	TrashcanBuf's length
in, out	<i>FilesetRoot</i>	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

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>HPSS_EFAULT</i>	Path or Trashcan Buffer not provided.
<i>HPSS_ENOENT</i>	Path does not exist
<i>HPSS_ENOTDIR</i>	Some component of the path is not a directory
<i>HPSS_EINVAL</i>	Invalid trashcan length specified or no handle provided.

Note

See `hpss_GetTrashcan` for additional notes about this function.

9.1.6.131 hpss_GetTrashSettings()

```
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	<i>TrashSettings</i>	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.

9.1.6.132 hpss_GetUID()

```
int32_t hpss_GetUID (
    uid_t * UID )
```

Get the user id for the calling thread.

\inuser client_api

The 'hpss_GetUID' function gets the current UID for the calling thread.

Return values

0	Success
<0	Error from API_ClientAPIInit() .

9.1.6.133 hpss_HashFlagsToString()

```
void hpss_HashFlagsToString (
    unsigned16 flags,
    char * buf,
    int len )
```

Returns a string representation of the provided hash flags.

Parameters

in	<i>flags</i>	Hash flags
in, out	<i>buf</i>	String buffer
in	<i>len</i>	String buffer length

Returns

char *

- String representation of the hash flags

9.1.6.134 hpss_InsertDistFile()

```
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	<i>FileInfo</i>	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 had previously been extracted using `hpss_GetDistFile`.

9.1.6.135 hpss_JunctionCreate()

```
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 filesset handle.

Parameters

in	<i>Path</i>	Path of the new junction
in	<i>SourceHandle</i>	Directory or filesset handle
out	<i>JunctionHandle</i>	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.

9.1.6.136 hpss_JunctionCreateHandle()

```
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 filesset handle.

The 'hpss_JunctionCreateHandle' function is called to create a HPSS junction to the specified directory or filesset handle.

Parameters

in	<i>ParentHandle</i>	directory to insert junction
in	<i>Path</i>	path name of the new junction
in	<i>SourceHandle</i>	Directory or filesset handle
in	<i>Ucred</i>	user credentials
out	<i>JunctionHandle</i>	Junction handle

Returns

The object handle for the new junction

Return values

<i>0</i>	Success
<i>-EFAULT</i>	Either the Path, SourceHandle, or JunctionHandle parameter is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system imposed limit, or a component of the pathname exceeds the system imposed limit.
<i>-ENOENT</i>	The Path argument points to an empty string.
<i>-EEXIST</i>	The named path already exists in the HPSS name space.
<i>-EACCES</i>	The requesting client is not the root user or a trusted user with write permissions.
<i>-EINVAL</i>	The SourceHandle parameter doesn't point to a directory handle or the Path is invalid.

9.1.6.137 hpss_JunctionDelete()

```
int hpss_JunctionDelete (
    const char * Path )
```

Deletes a junction.

The 'hpss_JunctionDelete' function is called to delete a junction.

Parameters

<i>in</i>	<i>Path</i>	Path of junction to delete

Return values

<i>0</i>	Success
<i>-ENOENT</i>	The Path parameter is an empty string or doesn't refer to an existing object.
<i>-EFAULT</i>	The Path parameter is NULL.
<i>-EINVAL</i>	The Path parameter doesn't refer to a junction.
<i>-EACCES</i>	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.

9.1.6.138 hpss_JunctionDeleteHandle()

```
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	<i>ParentHandle</i>	Parent directory
in	<i>Path</i>	Name of junction
in	<i>Ucred</i>	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.

9.1.6.139 hpss_Link()

```
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	<i>Existing</i>	Existing name of the object
in	<i>New</i>	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.

Return values

<i>-EPERM</i>	The object specified by Existing is a directory.
<i>-EMLINK</i>	The object specified by Existing is a directory.
<i>-ENAMETOOLONG</i>	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.
<i>-ENOENT</i>	No entry exists for the specified file.
<i>-ENOTDIR</i>	A component of the path prefix is not a directory.

9.1.6.140 **hpss_LinkHandle()**

```
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	<i>ObjHandle</i>	handle of existing obj
in	<i>DirHandle</i>	handle of parent directory
in	<i>New</i>	New name of the object
in	<i>Ucred</i>	user credentials

Return values

<i>0</i>	Link was successful.
<i>-EACCES</i>	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the new link.
<i>-EEXIST</i>	The object identified by New already exists.
<i>-EFAULT</i>	The ObjHandle, DirHandle, or New parameter is a NULL pointer.
<i>-EPERM</i>	The object specified by ObjHandle is a directory.
<i>-EMLINK</i>	The number of links to the file named by ObjHandle would exceed the system-imposed limit.
<i>-ENAMETOOLONG</i>	The length of the New argument exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
<i>-ENOENT</i>	New points to a null string.
<i>-ENOTDIR</i>	A component of the path prefix is not a directory.

9.1.6.141 hpss_LinkHandleParent()

```
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	<i>SrcDirHandle</i>	Handle to source parent directory
in	<i>SrcFile</i>	Path within source parent directory
in	<i>DestDirHandle</i>	Handle to destination parent directory
in	<i>DestFile</i>	New link path within destination parent directory
in	<i>Ucred</i>	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.

9.1.6.142 hpss_LoadDefaultThreadState()

```
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 'Client↔FullName' 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	<i>UserID</i>	New user credentials
in	<i>Umask</i>	New umask
in	<i>ClientFullName</i>	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.

9.1.6.143 hpss_LoadThreadState()

```
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

<i>UserID</i>	New user credentials
<i>Umask</i>	New umask
<i>ClientFullName</i>	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

9.1.6.144 hpss_LocateRootCS()

```
int32_t hpss_LocateRootCS (
    hpss_reqid_t RequestId,
    ls_map_t * Location )
```

Find the local root Core Server.

Parameters

in	<i>RequestId</i>	Id of the request to use for logging
out	<i>Location</i>	Returned map information about the root Core Server

Return values

0	Successful return
-EFAULT	Location is NULL
-ECONN	A communication error has occurred
-ENOENT	No information exists for the local root CS
-ENOMEM	Out of memory
-EMutex	An internal error occurred while accessing a mutex
HPSS_ESYSTEM	An internal error has occurred.
Otherwise	Some other error has occurred

9.1.6.145 hpss_LocateServerByID()

```
int32_t hpss_LocateServerByID (
    hpss_reqid_t RequestId,
    hpss_srvr_id_t ServerId,
    ls_map_t * Location )
```

Locate a server by its Id This routine locates a server by its unique ID. If ServerId is zero (Invalid), information about the local root Core Server is returned.

Parameters

in	<i>RequestId</i>	Id of this request to use when logging
in	<i>ServerId</i>	Id of the server to look up. If this is empty, information about the local root CS is returned.
out	<i>Location</i>	Returned map information about the server

Return values

0	Successful return
HPSS_EFAULT	Location is NULL

Return values

<i>HPSS_ECONN</i>	A communication error has occurred
<i>HPSS_ENOENT</i>	No information exists for the specified server
<i>HPSS_ENOMEM</i>	Out of memory
<i>HPSS_EMUTEX</i>	An internal error occurred while accessing a mutex
<i>HPSS_ESYSTEM</i>	An internal error has occurred.
<i>Otherwise</i>	Some other error has occurred

9.1.6.146 **hpss_LocateSiteById()**

```
int32_t hpss_LocateSiteById (
    hpss_reqid_t RequestId,
    hpss_id_t SiteId,
    char * SiteName )
```

Lookup a site's name by its id.

Parameters

in	<i>RequestId</i>	Id of this request to use when logging
in	<i>SiteId</i>	Id of the site to lookup information for. If this is NULL or points to a zero id then the local site is assumed.
out	<i>SiteName</i>	Returned name of the site. Max size is HPSS_MAX_SITE_NAME.

Return values

<i>0</i>	Successful return
<i>HPSS_EFAULT</i>	SiteName is NULL
<i>HPSS_ECONN</i>	A communication error has occurred
<i>HPSS_ENOENT</i>	No site information matches criteria
<i>HPSS_ENOMEM</i>	Out of memory
<i>HPSS_EMUTEX</i>	An internal error occurred while accessing a mutex
<i>HPSS_ESYSTEM</i>	An internal error has occurred
<i>Otherwise</i>	Some other error has occurred

9.1.6.147 **hpss_LocateSiteByName()**

```
int32_t hpss_LocateSiteByName (
    hpss_reqid_t RequestId,
    char * SiteName,
    hpss_id_t * SiteId )
```

Lookup a site's id by its name.

Parameters

in	<i>RequestId</i>	Id of this request to use when logging
in	<i>SiteName</i>	Name of the site to lookup. Max size is HPSS_MAX_SITE_NAME. If this is null the local site is assumed.
out	<i>SiteId</i>	Returned Id of the site found.

Return values

0	Successful return
HPSS_EFAULT	SiteId is NULL
HPSS_ECONN	A communication error has occurred
HPSS_ENOENT	No site information matches criteria
HPSS_ENOMEM	Out of memory
HPSS_EMUTEX	An internal error occurred while accessing a mutex
HPSS_ESYSTEM	An internal error has occurred
Otherwise	Some other error has occurred

9.1.6.148 hpss_LookupRootCS()

```
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	<i>SiteName</i>	HPSS site name
out	<i>ServerId</i>	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

9.1.6.149 hpss_Lseek()

```
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	<i>Fildes</i>	ID of open object
in	<i>Offset</i>	# of bytes to calculate new offset
in	<i>Whence</i>	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.

9.1.6.150 hpss_Lstat()

```
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	<i>Path</i>	path of file to get statistics
out	<i>Buf</i>	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.

Return values

<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.

9.1.6.151 hpss_Migrate()

```
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	<i>Fildes</i>	ID of open file/directory
in	<i>SrcLevel</i>	from where to migrate data
in	<i>Flags</i>	controlling flags
out	<i>RetBytesMigrated</i>	number of bytes migrated

Return values

0	Success
<i>-EBADF</i>	The supplied file descriptor does not correspond to a file opened for writing.
<i>-EBUSY</i>	The specified file descriptor is in use.
<i>-EFAULT</i>	The RetBytesMigrated parameter is a NULL pointer.
<i>-EINVAL</i>	The Flags argument is invalid.
<i>-EPERM</i>	The client does not have the appropriate privileges to issue explicit file migration requests.

9.1.6.152 hpss_MiscAdmin()

```
int hpss_MiscAdmin (
    uint32_t Function,
    char * TextP,
    ns_AdminConfArray_t * ValueConfArrayP )
```

Accesses the Core Server's miscellaneous functions.

Parameters

in	<i>Function</i>	Func to perform.
in, out	<i>TextP</i>	For auxil text.
in, out	<i>ValueConfArrayP</i>	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 logging 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 `ValueConfArrayP` is a conformant array of `uint64_t`'s and is used to pass any numerical data required or output by a `Function`.

Return values

<code>0</code>	No error. User List in buffer.
<code>-EINVAL</code>	The <code>SiteName</code> parameter is invalid
<code>-EFAULT</code>	The <code>ServerId</code> parameter is NULL
<code>-ENOENT</code>	The Root Core Server could not be located

9.1.6.153 hpss_Mkdir()

```
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	<i>Path</i>	path of directory
in	<i>Mode</i>	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.

9.1.6.154 hpss_MkdirHandle()

```
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 'RetVAttr', respectively.

Parameters

in	<i>ObjHandle</i>	handle of parent directory

Parameters

in	<i>Path</i>	path of directory
in	<i>Mode</i>	perm bits for new directory
in	<i>Ucred</i>	user credentials
out	<i>HandleOut</i>	returned object handle
out	<i>AttrOut</i>	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.

9.1.6.155 hpss_Open()

```
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	<i>Path</i>	Names the file to be opened or created.
in	<i>Oflag</i>	Specifies the file status and access modes to be assigned. Applicable values give 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	<i>Mode</i>	Specifies the file mode for a file that is created as a result of O_CREAT.

Parameters

in	<i>HintsIn</i>	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	<i>HintsPri</i>	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	<i>HintsOut</i>	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.
<i>-EACCES</i>	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.
<i>-EEXIST</i>	O_CREAT and O_EXCL are set and the named file exists.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINPROGRESS</i>	The file is currently being staged. The open should be retried at a later time.
<i>-EINVAL</i>	Oflag is not valid, or one or more values input in the HintsIn parameter is invalid.
<i>-EISDIR</i>	The named file is a directory. Note that opening directories via hpss_Open is not supported in any mode.
<i>-EMFILE</i>	The client open file table is already full.
<i>-ENFILE</i>	Too many files are open in the system.
<i>-ENAMETOOLONG</i>	The length of the Path string exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist and the O_CREAT flag was not specified, or the Path argument points to an empty string.
<i>-EREMOTE</i>	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.

9.1.6.156 hpss_OpenBitfile()

```
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	<i>BitfileObj</i>	Bitfile Object Handle (usually obtained using <code>hpss_FileGetAttributes</code>)
in	<i>Oflag</i>	Specifies file status and file access modes to be assigned. Applicable values below may be OR'd together. • <code>O_RDONLY</code> • <code>O_WRONLY</code> • <code>O_RDWR</code> • <code>O_APPEND</code> • <code>O_TRUNC</code> • <code>HPSS_O_NO_TAPE</code> • <code>HPSS_O_EXCL</code> • <code>HPSS_O_STAGE_ASYNC</code> • <code>HPSS_O_STAGE_BKGRD</code>
in	<i>Ucred</i>	User Credentials

Return values

<code>-EACCESS</code>	The client does not have permission for the requested file access.
<code>-EFAULT</code>	The BitFileId parameter is a NULL pointer.
<code>-EINPROGRESS</code>	The file is currently being staged. The open should be retried at a later time.
<code>-EINVAL</code>	Oflag is not valid.
<code>-EMFILE</code>	The client file table is already full.
<code>-ENFILE</code>	Too many bitfiles are already open in the system.
<code>-ENOENT</code>	No entry exists for the specified bitfile id.
<code>-EREMOTE</code>	The caller requested the <code>HPSS_O_NO_TAPE</code> 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.

9.1.6.157 hpss_OpenBitfileVAttr()

```
int hpss_OpenBitfileVAttr (
    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 opening 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.

9.1.6.158 hpss_Opendir()

```
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	<i>DirName</i>	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.

9.1.6.159 hpss_OpendirHandle()

```
int hpss_OpendirHandle (
    const ns_ObjHandle_t * DirHandle,
    const sec_cred_t * Ucred )
```

Open a directory via a handle.

Parameters

in	<i>DirHandle</i>	Directory Handle
in	<i>Ucred</i>	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.

9.1.6.160 hpss_OpenHandle()

```
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	<i>ObjHandle</i>	Parent object handle; the parent directory.
in	<i>Path</i>	Names the file to be opened or created.
in	<i>Oflag</i>	Specifies the file status and access modes to be assigned. Applicable values give 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	<i>Ucred</i>	User Credentials
in	<i>Mode</i>	Specifies the file mode for a file that is created as a result of O_CREAT.
in	<i>Ucred</i>	User credentials
in	<i>HintsIn</i>	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	<i>HintsPri</i>	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	<i>HintsOut</i>	The hints which were used for this file open; note that these might differ from the hints provided.
out	<i>AttrsOut</i>	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

<i>-EACCESS</i>	One of the following occurred: - Search permission is denied on a component of the path prefix. - The file exists and the permissions specified by <i>Oflag</i> are denied. - The file does not exist and write permission is denied for the parent directory of the file to be created. <i>O_TRUNC</i> is specified and write permission is denied.
<i>-EEXIST</i>	<i>O_CREAT</i> and <i>O_EXCL</i> are set and the named file exists.
<i>-EFAULT</i>	The <i>Path</i> parameter is a NULL pointer.
<i>-EINPROGRESS</i>	The file is currently being staged. The open should be retried at a later time.
<i>-EINVAL</i>	<i>Oflag</i> is not valid, or one or more values input in <i>HintsIn</i> are invalid.
<i>-EISDIR</i>	The named file is a directory. Note that opening directories via this function is not supported in any mode.
<i>-EMFILE</i>	The client open file table is already full.
<i>-ENFILE</i>	Too many open files on the system.
<i>-ENAMETOOLONG</i>	The length of the <i>Path</i> string, or a path name component exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist and the <i>O_CREAT</i> flag was not specified, or the <i>Path</i> argument points to an empty string.
<i>-ENOSPC</i>	Resources could not be allocated for the new file.
<i>-ENOTDIR</i>	A component of the path is not a directory.
<i>-EREMOTE</i>	The caller requested the <i>HPSS_O_NO_TAPE</i> 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.

9.1.6.161 hpss_PingCore()

```
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	<i>CoreServerID</i>	Core Server ID
out	<i>RecvSecs</i>	Secs received
out	<i>RecvUSEcs</i>	USEcs received

Return values

0	Success
-EFAULT	Null parameter
-ENOCONECT	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.

9.1.6.162 hpss_PIOEnd()

```
int hpss_PIOEnd (
    hpss_pio_grp_t StripeGroup )
```

End and clean up a parallel IO group context.

Parameters

in, out	<i>StripeGroup</i>	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.

9.1.6.163 hpss_PIOExecute()

```
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	<i>Fd</i>	Open file descriptor
in	<i>FileOffset</i>	IO file offset
in	<i>Size</i>	IO size
in	<i>StripeGroup</i>	IO stripe group
out	<i>GapInfo</i>	Sparse file information
out	<i>BytesMoved</i>	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.

9.1.6.164 hpss_PIOExecuteItem()

```
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	<i>Fd</i>	Open file descriptor
in	<i>FileOffset</i>	IO file offset
in	<i>Size</i>	IO size
in	<i>StripeGroup</i>	IO stripe group
out	<i>GapInfo</i>	Sparse file information
out	<i>BytesMoved</i>	Bytes moved
in	<i>Item</i>	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.

9.1.6.165 hpss_PIOExportGrp()

```
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	<i>StripeGroup</i>	IO stripe group
out	<i>Buffer</i>	Raw data buffer
out	<i>BufLength</i>	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.

9.1.6.166 hpss_PIOFinalize()

```
int hpss_PIOFinalize (
    hpss_pio_grp_t * StripeGroup )
```

Safely end and clean up a parallel IO group context.

Parameters

in	<i>StripeGroup</i>	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

<i>-EFAULT</i>	NULL stripe group provided
<i>0</i>	Success
<i>Other</i>	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.

9.1.6.167 hpss_PIOGetPartError()

```
int hpss_PIOGetPartError (
    hpss_pio_grp_t StripeGroup,
    int * Error )
```

Retrieves a participant error code.

Parameters

in	<i>StripeGroup</i>	IO stripe group
in, out	<i>Error</i>	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

9.1.6.168 hpss_PIOGetRequestID()

```
int hpss_PIOGetRequestID (
    hpss_pio_grp_t StripeGroup,
    hpss_reqid_t * RequestID )
```

Retrieves the request ID for a group.

Parameters

in	<i>StripeGroup</i>	IO stripe group
in, out	<i>RequestID</i>	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.

9.1.6.169 hpss_PIOImportGrp()

```
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	<i>Buffer</i>	Raw data buffer
in	<i>BufLength</i>	Raw buffer length
out	<i>StripeGroup</i>	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

<i>-ENOMEM</i>	Can't allocate memory for stack entry
<i>-EINVAL</i>	StripeGroup or Buffer is NULL
0	otherwise

Note

This function runs in the participant context/role.

9.1.6.170 hpss_PIOInitCtx()

```
int hpss_PIOInitCtx (
    hpss_pio_t * p )
```

Start a simplified PIO context.

Parameters

<i>in, out</i>	<i>Ctx</i>	Context to modify

This creates a PIO context which can be used with [hpss_PIOReadFileMulti](#) and [hpss_PIOWriteFileMulti](#). The PIO options can be set using [hpss_PIOSetStripeWidth](#), [hpss_PIOSetOffset](#), and [hpss_PIOSetDefaultCtx](#).

Return values

0	Success
<i>-EINVAL</i>	Invalid input parameters

Note

The PIO context must be reset (using [hpss_PIOResetCtx](#)) between calls.

9.1.6.171 hpss_PIOReadFile()

```
int hpss_PIOReadFile (
    int hpss_fd,
    uint64_t data_length,
    int local_fd,
    uint64_t * bytes_copied,
    hpss_read_queue_item_t * item )
```

Read a file from HPSS to a local file using PIO.

Parameters

in	<i>hpss_fd</i>	Open HPSS File Descriptor
in	<i>data_length</i>	Amount of data to read from HPSS
in	<i>local_fd</i>	Open local file descriptor to write to
out	<i>bytes_copied</i>	Number of bytes copied to local_fd
in	<i>item</i>	Read Queue Item (optional)

This function will use PIO with default settings (from `hpss_PIOSetDefaultCtx`) to copy data from the `hpss_fd` to `local_fd`, up to `data_length` bytes. If `item` is provided, it will be used to take advantage of read queuing.

Return values

0	Success
-EIO	I/O Error
-EINVAL	Invalid input parameters

9.1.6.172 `hpss_PIOReadFileMulti()`

```
int hpss_PIOReadFileMulti (
    hpss_pio_t * Ctx,
    int hpss_fd,
    uint64_t data_length,
    int local_fd,
    uint64_t * bytes_copied,
    hpss_read_queue_item_t * item )
```

Issue a simplified PIO read.

Parameters

in, out	<i>Ctx</i>	Context to use
in	<i>hpss_fd</i>	Open HPSS File Descriptor
in	<i>data_length</i>	Amount of data to read from HPSS
in	<i>local_fd</i>	Open local file descriptor to write to
out	<i>bytes_copied</i>	Number of bytes copied to local_fd
in	<i>item</i>	Read Queue Item (optional)

This function will use PIO with default settings (from `hpss_PIOSetDefaultCtx`) to copy data from the `hpss_fd` to `local_fd`, up to `data_length` bytes. If `item` is provided, it will be used to take advantage of read queuing.

Return values

0	Success
-EIO	I/O Error
-EINVAL	Invalid input parameters

9.1.6.173 hpss_PIORegister()

```
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	<i>StripeElement</i>	Element of the stripe
in	<i>DataNetSockAddr</i>	NIC address for data movement
in	<i>DataBuffer</i>	Data buffer for IO
in	<i>DataBufLen</i>	Data buffer length
in, out	<i>StripeGroup</i>	IO stripe group
in	<i>IOCallback</i>	IO call back function
in	<i>IOCallbackArg</i>	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.

9.1.6.174 hpss_PIOResetCtx()

```
int hpss_PIOResetCtx (
    hpss_pio_t * p )
```

Stop a simplified PIO context.

Parameters

in, out	<i>Ctx</i>	Context to modify

This resets a PIO context. This allows it to be used again with `hpss_PIOInitCtx` or a call to `hpss_PIOReadFileMulti` and `hpss_PIOWriteFileMulti`.

Return values

0	Success
-EINVAL	Invalid input parameters

9.1.6.175 `hpss_PIOSetBlockSize()`

```
int hpss_PIOSetBlockSize (
    hpss_pio_t * Ctx,
    uint32_t BlockSize )
```

Set the client transfer block size.

Parameters

in, out	<i>Ctx</i>	Context to modify
in	<i>Width</i>	Client stripe width

Update the block size for the context to the provided value.

Return values

0	Success
-EINVAL	Invalid input parameters

9.1.6.176 `hpss_PIOSetDefaultCtx()`

```
int hpss_PIOSetDefaultCtx (
    hpss_pio_t * Ctx )
```

Set default values in the PIO context.

Parameters

in, out	<i>Ctx</i>	Context to fill with defaults

This function applies any default values to the input context. Only values which have environment defaults configured will be changed.

This function responds to the following environment variables:

- `HPSS_API_PIO_STRIPE_WIDTH` (int), stripe width (must be $\leq$ `HPSS_MAX_STRIPE_WIDTH`)

- `HPSS_API_PIO_BLOCK_SIZE` (int), transfer block size
- `HPSS_API_PIO_HANDLE_GAPS` (1/0), handle gaps
- `HPSS_API_PIO_USE_API_HOST` (1/0), use the `HPSS_API_HOSTNAME` for the data hostname for Mover connections

Return values

0	Success

9.1.6.177 `hpss_PIOSetHandleGaps()`

```
int hpss_PIOSetHandleGaps (
    hpss_pio_t * Ctx,
    int HandleGaps )
```

Set the client gap handling behavior.

Parameters

in, out	<i>Ctx</i>	Context to modify
in	<i>HandleGaps</i>	Handle Gaps (true/false)

Update the gap handling behavior for the context to the provided value.

Return values

0	Success
-EINVAL	Invalid input parameters

9.1.6.178 `hpss_PIOSetStripeWidth()`

```
int hpss_PIOSetStripeWidth (
    hpss_pio_t * Ctx,
    uint32_t Width )
```

Set the client stripe width.

Parameters

in, out	<i>Ctx</i>	Context to modify
in	<i>Width</i>	Client stripe width

Update the stripe width for the context to the provided value.

Return values

0	Success
-EINVAL	Invalid input parameters

9.1.6.179 hpss_PIOSetUseAPIHost()

```
int hpss_PIOSetUseAPIHost (
    hpss_pio_t * Ctx,
    int UseAPIHost )
```

Set the client data hostname behavior.

Parameters

in, out	<i>Ctx</i>	Context to modify
in	<i>UseAPIHost</i>	Use API Host (true/false)

Update the API host

Return values

0	Success
-EINVAL	Invalid input parameters

9.1.6.180 hpss_PIOStart()

```
int hpss_PIOStart (
    hpss_pio_params_t * InputParams,
    hpss_pio_grp_t * StripeGroup )
```

Start a new parallel I/O group context.

Parameters

in, out	<i>InputParams</i>	Input parameters
out	<i>StripeGroup</i>	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.

Return values

-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.

9.1.6.181 hpss_PIOWriteFile()

```
int hpss_PIOWriteFile (
    int hpss_fd,
    uint64_t data_length,
    int local_fd,
    uint64_t * bytes_copied )
```

Write a file to HPSS from a local file using PIO.

Parameters

in	<i>hpss_fd</i>	Open HPSS File Descriptor
in	<i>data_length</i>	Amount of data to write to HPSS
in	<i>local_fd</i>	Open local file descriptor to read from
out	<i>bytes_copied</i>	Number of bytes copied to hpss_fd

This function will use PIO with default settings (from *hpss_PIOSetDefaultCtx*) to copy data from the *local_fd* to *hpss_fd*, up to *data_length* bytes.

Return values

0	Success
-EIO	I/O Error
-EINVAL	Invalid input parameters

9.1.6.182 hpss_PIOWriteFileMulti()

```
int hpss_PIOWriteFileMulti (
    hpss_pio_t * Ctx,
    int hpss_fd,
    uint64_t data_length,
    int local_fd,
    uint64_t * bytes_copied )
```

Write a file to HPSS from a local file using PIO.

Parameters

in, out	<i>Ctx</i>	Context to use
in	<i>hpss_fd</i>	Open HPSS File Descriptor
in	<i>data_length</i>	Amount of data to write to HPSS
in	<i>local_fd</i>	Open local file descriptor to read from
out	<i>bytes_copied</i>	Number of bytes copied to <i>hpss_fd</i>

This function will use PIO with default settings (from `hpss_PIOSetDefaultCtx`) to copy data from the *local_fd* to *hpss_fd*, up to *data_length* bytes.

Return values

0	Success
-EIO	I/O Error
-EINVAL	Invalid input parameters

9.1.6.183 `hpss_PREFERREDReadOrder()`

```
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	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read context to use
in	<i>UserCredsP</i>	User Credentials
in	<i>ItemsToCheck</i>	List of read requests to check on (or NULL)
in	<i>Flags</i>	Flags • <code>CORE_RQ_ONLY_NEW_READY</code>. Only return new, ready read requests.
in	<i>Delay</i>	How long to check for requests which are ready to run
out	<i>ItemsReady</i>	Which items are ready to run
out	<i>ItemsError</i>	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

<i>0</i>	Success
<i>-EFAULT</i>	One or more of ContextP, ItemsReady, or ItemsError is NULL
<i>-ENOENT</i>	ItemsToCheck has no entries, or nothing is ready
<i>Other</i>	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.

9.1.6.184 `hpss_Purge()`

```
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	<i>Fildes</i>	ID of open file/directory
in	<i>Offset</i>	beginning offset
in	<i>Length</i>	length to be purged
in	<i>StorageLevel</i>	where to purge data
in	<i>Flags</i>	controlling flags
out	<i>RetBytesPurged</i>	number of bytes purged

Return values

<i>0</i>	No error.
<i>-EBADF</i>	The supplied file descriptor does not correspond to an open file.
<i>-EBUSY</i>	The specified file descriptor is in use.
<i>-EFAULT</i>	The RetBytesPurged parameter is a NULL pointer.
<i>-EINVAL</i>	The Flags, Offset or Length argument is invalid.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform explicit purge operations.

9.1.6.185 hpss_PurgeLock()

```
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	<i>Fildes</i>	ID of open object
in	<i>Flag</i>	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.

9.1.6.186 hpss_PurgeLoginCred()

```
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.

9.1.6.187 hpss_PurgeOnMigrate()

```
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	<i>Fildes</i>	Open file handle
in	<i>Flag</i>	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.

9.1.6.188 hpss_Read()

```
ssize_t hpss_Read (
    int Fildes,
    void * Buf,
    size_t Nbyte )
```

Reads a number of bytes from a file.

Parameters

in	<i>Fildes</i>	ID of object to be read
in	<i>Buf</i>	Buffer in which to receive data
in	<i>Nbyte</i>	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

<i>positive</i>	Number of Bytes Read
<i>0</i>	End of file reached
<i>-EBADF</i>	The specified file descriptor does not correspond to a file opened for reading.
<i>-EBUSY</i>	The file is currently in use by another client thread.
<i>-EFAULT</i>	The Buf parameter is NULL.
<i>-EIO</i>	An input/output or HPSS internal error occurred.

9.1.6.189 hpss_ReadAttrs()

```
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	<i>Dirdes</i>	open directory stream handle
in	<i>OffsetIn</i>	directory position
in	<i>BufferSize</i>	size of output buffer
in	<i>GetAttributes</i>	get object attributes
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	directory entry information

Return values

<i>0</i>	No error. 'DirentPtr' contains directory information.
<i>-EBADF</i>	The specified directory descriptor does not refer to an open directory.
<i>-EBUSY</i>	The directory is currently in use by another client thread.
<i>-EFAULT</i>	The DirentPtr, End or OffsetOut parameter is a NULL pointer.
<i>-EINVAL</i>	The BufferSize parameter is zero.

Deprecated This function will be removed in a future version of HPSS. Use [hpss_ReadAttrsPlus\(\)](#).

9.1.6.190 hpss_ReadAttrsHandle()

```
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	<i>ObjHandle</i>	directory object handle
in	<i>OffsetIn</i>	directory position
in	<i>Ucred</i>	user credentials
in	<i>BufferSize</i>	size of output buffer
in	<i>GetAttributes</i>	get object attributes
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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\(\)](#).

9.1.6.191 hpss_ReadAttrsPlus()

```
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	<i>Dirdes</i>	open directory stream handle
in	<i>OffsetIn</i>	directory position
in	<i>BufferSize</i>	size of output buffer
in	<i>Flags</i>	readdir flags
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	directory entry information

Return values

>0	Number of 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.

9.1.6.192 hpss_Readdir()

```

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	<i>Dirdes</i>	open directory stream handle
out	<i>DirentPtr</i>	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 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.

9.1.6.193 hpss_ReaddirHandle()

```
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	<i>ObjHandle</i>	directory object handle
in	<i>OffsetIn</i>	directory position
in	<i>Ucred</i>	user credentials
in	<i>BufferSize</i>	size of output buffer
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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\(\)](#).

9.1.6.194 hpss_ReaddirPlus()

```
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	<i>Dirdes</i>	Open directory descriptor
in	<i>OffsetIn</i>	directory position
in	<i>BufferSize</i>	size of output buffer
in	<i>Flags</i>	readdir flags
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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.

9.1.6.195 hpss_ReadItem()

```
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	<i>Fildes</i>	ID of object to be read
in	<i>Buf</i>	Buffer in which to receive data
in	<i>Nbyte</i>	number of bytes to read
in	<i>Item</i>	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

<i>positive</i>	Number of Bytes Read
<i>0</i>	End of file reached
<i>-EBADF</i>	The specified file descriptor does not correspond to a file opened for reading.
<i>-EBUSY</i>	The file is currently in use by another client thread.
<i>-EFAULT</i>	The Buf or Item parameter is NULL.
<i>-EIO</i>	An input/output or HPSS internal error occurred.
<i>-EIO</i>	Failed to set the request ID
<i>Other</i>	POSIX error indicating the nature of the error

9.1.6.196 hpss_Readlink()

```
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	<i>Path</i>	Name of the link
out	<i>Contents</i>	Contents of the link
in	<i>BufferSize</i>	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.

9.1.6.197 hpss_ReadlinkHandle()

```
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	<i>ObjHandle</i>	handle of parent
in	<i>Path</i>	Name of symlink
in	<i>BufferSize</i>	Size, in bytes, of Contents
out	<i>Contents</i>	Contents of the link
in	<i>Ucred</i>	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.

Return values

<i>-EINVAL</i>	The specified file is not a symbolic link.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The specified path name does not exist.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-ERANGE</i>	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.

9.1.6.198 hpss_ReadList()

```
int hpss_ReadList (
    const hpss_IOD_t * IODPtr,
    uint32_t Flags,
    hpss_IOR_t * IORPtr )
```

Send a read IOD to HPSS.

Parameters

in	<i>IODPtr</i>	IO description
in	<i>Flags</i>	IO specific flags
out	<i>IORPtr</i>	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.

9.1.6.199 hpss_ReadQueueAdd()

```
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	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read context to add to
in	<i>Item</i>	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

9.1.6.200 hpss_ReadQueueAddMany()

```
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	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read context to add to
in	<i>ItemList</i>	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

9.1.6.201 hpss_ReadQueueCreateContext()

```
signed32 hpss_ReadQueueCreateContext (
    uint32_t SubsysId,
    sec_cred_t * UserCredsP,
    hpss_uuid_t * ContextP )
```

Create a new read queue context.

Parameters

in	<i>SubsysId</i>	Subsystem ID
out	<i>ContextP</i>	Context UUID

Return values

0	Success
-EFAULT	ContextP is NULL
Other	POSIX error indicating the nature of the error

9.1.6.202 hpss_ReadQueueList()

```
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	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read context to use
in	<i>UserCredsP</i>	User Credentials
in	<i>VVID</i>	VV to List (optional)
in	<i>Start</i>	Initial list index to start listing from in ContextP
in	<i>Max</i>	Maximum number of items to return
out	<i>List</i>	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

9.1.6.203 hpss_ReadQueueRemove()

```
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	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read context to use
in	<i>UserCredsP</i>	User Credentials
in	<i>RequestList</i>	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

9.1.6.204 hpss_ReadQueueRemoveContext()

```

signed32 hpss_ReadQueueRemoveContext (
    uint32_t SubsysId,
    hpss_uuid_t * ContextP,
    sec_cred_t * UserCredsP,
    int Flags )

```

Remove a read queue context.

Parameters

in	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read queue context to use
in	<i>Flags</i>	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

9.1.6.205 hpss_ReadRawAttrsHandle()

```
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	<i>ObjHandle</i>	directory object handle
in	<i>OffsetIn</i>	directory position
in	<i>Ucred</i>	user credentials
in	<i>BufferSize</i>	size of output buffer
in	<i>GetAttributes</i>	get object attributes
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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\(\)](#).

9.1.6.206 hpss_RemoveIntersectionFromList()

```
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	<i>List</i>	List of all items to process
in	<i>Partial</i>	Sublist of items to remove

Returns

A new allocation of list items

Note

The new list array must be freed

9.1.6.207 **hpss_Rename()**

```
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	<i>Old</i>	Old name of the object
in	<i>New</i>	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 dotdot.

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 the deleting user and its location.

9.1.6.208 hpss_RenameHandle()

```
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	<i>OldHandle</i>	Old parent object handle
in	<i>OldPath</i>	Old name of the object
in	<i>NewHandle</i>	New parent object handle
in	<i>NewPath</i>	New name of the object
in	<i>Ucred</i>	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 the deleting user and its location.

9.1.6.209 hpss_ReopenBitfile()

```
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	<i>Fildes</i>	Descriptor to be reopened
in	<i>BitfileObj</i>	Bitfile Object to Open
in	<i>Oflag</i>	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	<i>Ucred</i>	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.

Return values

<code>-ENOENT</code>	No entry exists for the specified bitfile id.
<code>-EREMOTE</code>	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.

9.1.6.210 hpss_ResetSubSysStats()

```
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	<i>SubsystemID</i>	identifier for the subsystem
out	<i>StatsOut</i>	subsystem statistics information

Return values

<code>0</code>	No error.
<code>-EFAULT</code>	The StatsOut parameter is a NULL pointer.

9.1.6.211 hpss_Rewinddir()

```
int hpss_Rewinddir (
    int Dirdes )
```

Rewind a directory stream back to the beginning.

Parameters

in	<i>Dirdes</i>	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.

9.1.6.212 hpss_Rmdir()

```
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 be overwritten by the rename, and be moved to the trashcan corresponding to the deleting user and the Path's location.

9.1.6.213 hpss_RmdirHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of directory
in	<i>Ucred</i>	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 be overwritten by the rename, and be moved to the trashcan corresponding to the deleting user and the Path's location.

9.1.6.214 hpss_RmdirHandleImmediate()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of directory
in	<i>Ucred</i>	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.

9.1.6.215 hpss_RmdirImmediate()

```
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.

9.1.6.216 hpss_SetAcct()

```
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	<i>NewCurAcct</i>	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()

9.1.6.217 hpss_SetAcctByName()

```
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	<i>NewAcctName</i>	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.

Return values

<i>-ENOENT</i>	The account specified by NewAcctName doesn't exist.
<i>-EPERM</i>	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.
<i>< 0</i>	Error from API_ClientAPIInit()

9.1.6.218 hpss_SetACL()

```
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	<i>Path</i>	path of file
in	<i>Options</i>	bitmask of ACL options
in	<i>ACL</i>	ACL array to be set

Return values

<i>0</i>	No error.
<i>-EACCES</i>	Search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path or ACL parameter is a NULL pointer.
<i>-EINVAL</i>	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.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.

9.1.6.219 hpss_SetACLHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file
in	<i>Ucred</i>	user credentials
in	<i>Options</i>	bitmask of ACL options
in	<i>ACL</i>	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.

9.1.6.220 hpss_SetAPILogLevel()

```
int hpss_SetAPILogLevel (
    int LogLevel )
```

Modifies the Client API log level.

Parameters

in	<i>LogLevel</i>	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

9.1.6.221 **hpss_SetAPILogPath()**

```
int hpss_SetAPILogPath (
    const char * LogPath )
```

Modifies the Client API log path.

Parameters

in	<i>LogPath</i>	Client API Log Path

Return values

0	Success
<i>EACCES</i>	Requested access is not allowed
<i>EFAULT</i>	LogPath is outside addressable address space
<i>EISDIR</i>	LogPath refers to a directory
<i>ELOOP</i>	Too many symbolic links were encountered
<i>EMFILE</i>	The process already has the maximum number of files open.
<i>ENAMETOOLONG</i>	LogPath is too long.
<i>ENOSPC</i>	No space for the new logfile
<i>ENOTDIR</i>	A component used as a directory in LogPath was not a directory
<i>EOVERFLOW</i>	LogPath refers to a regular file that is too large to be opened
<i>EROFS</i>	LogPath refers to a file on a read-only file system
<i>ENOMEM</i>	Ran out of memory

9.1.6.222 **hpss_SetApplicationName()**

```
void hpss_SetApplicationName (
    const char * App )
```

Modifies the Client API application name.

Parameters

in	<i>App</i>	Application name string

9.1.6.223 hpss_SetAppLoginCred()

```
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	<i>AppName</i>	Application name
in	<i>PrincipalName</i>	Principal name
in	<i>Mechanism</i>	Security Mechanism
in	<i>CredType</i>	Type of creds needed
in	<i>AuthType</i>	Authenticator type
in	<i>Authenticator</i>	Authenticator

Return values

0	Success
<i>Otherwise</i>	An other HPSS errno indicating nature 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 credential. This function is not thread safe, and they should only be called within a single thread, or serialized using a mutex.

9.1.6.224 hpss_SetAttrHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file to set attributes
in	<i>Ucred</i>	user credentials
in	<i>SelFlagsIn</i>	indication of attrs to be set
in	<i>AttrIn</i>	new attribute values
out	<i>SelFlagsOut</i>	indication of attrs set
out	<i>AttrOut</i>	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.

9.1.6.225 hpss_SetAuditInfo()

```

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	<i>EndUserHostAddr</i>	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

9.1.6.226 hpss_SetConfiguration()

```
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	<i>Config↔ In</i>	Configuration parameters

Return values

0	Success
-EFAULT	The ConfigIn parameter is NULL
-EINVAL	Invalid configuration attribute value setting

9.1.6.227 hpss_SetCOSByHints()

```
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	<i>Fildes</i>	ID of object to be read
in	<i>Flags</i>	operation flags to use
in	<i>HintsPtr</i>	COS hints desired
in	<i>PrioPtr</i>	Priorities of COS hints
out	<i>COSPtr</i>	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.

9.1.6.228 hpss_SetFileNotrunc()

```
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	<i>Path</i>	Pathname of file
in	<i>Flag</i>	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.

9.1.6.229 hpss_SetFileOffset()

```
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 offsets (ala lseek()).

Parameters

in	<i>Fildes</i>	ID of open object
in	<i>OffsetIn</i>	# of bytes to calculate new offset
in	<i>Whence</i>	Origin for the seek
in	<i>Direction</i>	OffsetIn is either forward or backward
out	<i>OffsetOut</i>	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.

9.1.6.230 hpss_SetGID()

```
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	<i>NewCurGid</i>	new value for current GID

Return values

0	Success
-EINVAL	The NewGroupID parameter is invalid.
<0	Error from API_ClientAPIInit() .

9.1.6.231 hpss_SetLoginCred()

```
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	<i>PrincipalName</i>	Principal name
in	<i>Mechanism</i>	Security Mechanism
in	<i>CredType</i>	Type of creds needed
in	<i>AuthType</i>	Authenticator type
in	<i>Authenticator</i>	Authenticator

Return values

0	Success
Otherwise	An other HPSS errno indicating nature 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 credential. This function is not thread safe, and they should only be called within a single thread, or serialized using a mutex.

9.1.6.232 hpss_SetNextIORequestID()

```
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_↔GetUniqueRequestID` or from `hpss_ReadQueueAdd`.

Sets the next request id

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>EFAULT</i>	NULL Request ID provided
<i>EINVAL</i>	Invalid request ID provided

Note

This should not be called except with a request ID provided via the `hpss_ReadQueueAddItem` or `hpss_↔ReadQueueAddMany` APIs.

9.1.6.233 hpss_SiteIdToName()

```
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	<i>SiteId</i>	site id
out	<i>SiteName</i>	site name

Return values

<i>0</i>	SiteId was translated successfully.
<i>-ENOENT</i>	The value passed in SiteId does not correspond to a known HPSS site.

Return values

-ECONN	There was a communication problem during the translation.

9.1.6.234 hpss_SiteNameToId()

```
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	<i>SiteName</i>	site name
out	<i>SiteId</i>	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.

9.1.6.235 hpss_Stage()

```
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	<i>Fildes</i>	ID of open file/directory
in	<i>Offset</i>	beginning offset
in	<i>Length</i>	length to be staged
in	<i>StorageLevel</i>	where to stage data
in	<i>Flags</i>	controlling flags

Return values

<i>0</i>	Success
<i>-EBADF</i>	The supplied file descriptor does not correspond to an open file.
<i>-EINVAL</i>	The Flags, Offset or Length argument is invalid.
<i>-EBUSY</i>	The specified file descriptor is currently in use.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.
<i>-EREMOTE</i>	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)

9.1.6.236 **hpss_StageBatch()**

```
int hpss_StageBatch (
    hpss_stage_batch_t * Batch,
    hpss_stage_batch_status_t * Status )
```

Stage a batch of bitfiles.

Parameters

in	<i>Batch</i>	Batch Stage Information
out	<i>Status</i>	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

<i>0</i>	Requests successfully batched
<i>-EFAULT</i>	NULL Batch and/or Status Structure Provided
<i>-EBADF</i>	One or more file descriptors in the batch stage structure was invalid
<i>-EBUSY</i>	One or more of the file descriptors in the batch stage structure was busy
<i>-ESTALE</i>	One or more of the file descriptors in the batch stage structure is no longer valid
<i>-ERANGE</i>	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.
<i>-EREMOTE</i>	The caller supplied the BFS_NO_TAPE stage flag, and some part of the file would be staged from tape.
<i>Other</i>	One or more requests failed to stage; review the Status structure for more information.

9.1.6.237 hpss_StageBatchCallBack()

```
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	<i>Batch</i>	Batch Stage Information
in	<i>CallBackPtr</i>	Callback Pointer
out	<i>ReqID</i>	Request Id
out	<i>BFObjsList</i>	Stage Bitfile Objects
out	<i>Status</i>	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.

Note that if a callback is provided and the Flags specify `BFS_NO_CB`, then the status may be checked with `hpss_GetCallbackStatus` or `hpss_GetAsyncStatus`, but no result will be sent to the callback socket. In this mode of operation a callback must be provided to allow the status of the batch to be checked.

Return values

<i>0</i>	Requests successfully batched
<i>-EFAULT</i>	NULL Request, Batch, Status or Bitfile ID Structure Provided
<i>-EINVAL</i>	Invalid Batch Structure Length Provided
<i>-ERANGE</i>	The size of the batch is larger than the legal size. Split the batch into multiple batches of <code>HPSS_MAX_STAGE_BATCH_LEN</code> or smaller and try again.
<i>Other</i>	One or more requests failed to stage; review the Status structure for more information.

Note

The callback id can be used to get the status of the stage requests using the `hpss_GetCallbackStatus` API. This function can be called multiple times with the same callback information to add entries onto the same callback over time. Because incoming entries are persisted, it is best to add entries in the 100s to 1000 at a time if entries are being added over time.

9.1.6.238 hpss_StageCallBack()

```
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 parameters can be provided to use as setup for later notification.

Note that if a callback is provided and the Flags specify BFS_NO_CB, then the status may be checked with `hpss_GetCallbackStatus` or `hpss_GetAsyncStatus`, but no result will be sent to the callback socket. In this mode of operation a callback must be provided to allow the status of the batch to be checked.

If BFS_NO_CB is not provided, the callback address must be resolvable by the HPSS core server or else the request will be rejected.

Parameters

in	<i>Path</i>	path to the object
in	<i>Offset</i>	beginning offset
in	<i>Length</i>	length to be staged
in	<i>StorageLevel</i>	where to stage data
in, out	<i>CallbackPtr</i>	callback information
in	<i>Flags</i>	controlling flags
out	<i>ReqID</i>	assigned request code
out	<i>BitfileObj</i>	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.

The callback id can be used to get the status of the stage requests using the `hpss_GetCallbackStatus` API.

This function can be called multiple times with the same callback information to add entries onto the same callback over time. Because incoming entries are persisted, it is best to add entries in the 100s to 1000 at a time if entries are being added over time.

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)
- BFS_NO_CB (Do not report result to the provided callback)

9.1.6.239 hpss_StageCallBackBitfile()

```
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.

Note that if a callback is provided and the Flags specify BFS_NO_CB, then the status may be checked with `hpss_↔_GetCallbackStatus` or `hpss_GetAsyncStatus`, but no result will be sent to the callback socket. In this mode of operation a callback must be provided to allow the status of the batch to be checked.

If BFS_NO_CB is not provided, the callback address must be resolvable by the HPSS core server or else the request will be rejected.

Parameters

in	<i>BitfileObj</i>	bitfile object
in	<i>Offset</i>	beginning offset
in	<i>Length</i>	length to be staged
in	<i>StorageLevel</i>	where to stage data
in, out	<i>CallBackPtr</i>	callback info
in	<i>Flags</i>	controlling flags
out	<i>ReqID</i>	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.

The callback id can be used to get the status of the stage requests using the `hpss_GetCallbackStatus` API.

This function can be called multiple times with the same callback information to add entries onto the same callback over time. Because incoming entries are persisted, it is best to add entries in the 100s to 1000 at a time if entries are being added over time.

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)
- `BFS_NO_CB` (Do not report result to the provided callback)

9.1.6.240 `hpss_Stat()`

```
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	<i>Path</i>	path of file to get statistics
out	<i>Buf</i>	Returned statistics

Return values

<code>0</code>	No error. Statistics in buffer.
<code>-EACCES</code>	Search permission is denied for a component of the path prefix.
<code>-EFAULT</code>	The Path or Buf parameter is a NULL pointer.
<code>-ENAMETOOLONG</code>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<code>-ENOENT</code>	The named file does not exist, or the Path argument points to an empty string.
<code>-ENOTDIR</code>	A component of the Path prefix is not a directory.

9.1.6.241 `hpss_Statfs()`

```
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 LS 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	<i>CosId</i>	Class of service id.
out	<i>StatfsBuffer</i>	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\(\)](#).

9.1.6.242 hpss_Statfs64()

```
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 LS 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	<i>CosId</i>	Class of service id.
out	<i>StatfsBuffer</i>	file system status.

Return values

0	Success
-EFAULT	The StatfsBuffer parameter is a NULL pointer.
-EINVAL	The specified Class of Service does not exist.

9.1.6.243 hpss_Statvfs()

```
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 LS 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	<i>CosId</i>	Class of service id.
out	<i>StatvfsBuffer</i>	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.

9.1.6.244 hpss_Statvfs64()

```
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 LS 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	<i>CosId</i>	Class of service id.
out	<i>StatvfsBuffer</i>	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.

9.1.6.245 hpss_Symlink()

```
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	<i>Contents</i>	Desired contents of the new link
in	<i>Path</i>	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.

9.1.6.246 hpss_SymlinkHandle()

```
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	<i>ObjHandle</i>	handle of parent
in	<i>Contents</i>	Contents of the link
in	<i>Path</i>	New name of symlink
in	<i>Ucred</i>	user credentials
out	<i>AttrsOut</i>	symbolic link attributes

Return values

<i>0</i>	Creation of symbolic link was successful.
<i>-EACCES</i>	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.
<i>-EFAULT</i>	The Path or Contents parameter is a NULL pointer.
<i>-EEXIST</i>	The specified link already exists.
<i>-EINVAL</i>	ObjHandle is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

9.1.6.247 hpss_ThreadCleanUp()

```
int hpss_ThreadCleanUp (
    void )
```

Cleans up a thread's context.

This routine is called to clean up a thread's context.

Return values

<i>0</i>	Success

9.1.6.248 hpss_Truncate()

```
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	<i>Path</i>	Pathname of file
in	<i>Length</i>	New length

Return values

<i>0</i>	Success
<i>-EACCES</i>	Search permission is denied on a component of the path prefix, or write permission is denied on the file.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.

Return values

<i>-EINVAL</i>	Path specifies a directory.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The specified path name does not exist.
<i>-ENOTDIR</i>	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.

9.1.6.249 `hpss_TruncateHandle()`

```
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	<i>ObjHandle</i>	Handle to file
in	<i>Path</i>	file name
in	<i>Length</i>	New length
in	<i>Ucred</i>	user credentials

Return values

0	Success
<i>-EACCES</i>	Search permission is denied on a component of the path prefix, or write permission is denied on the file.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	Path specifies a directory or ObjHandle is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The specified path name does not exist.
<i>-ENOTDIR</i>	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.

9.1.6.250 hpss_Umask()

```
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	<i>CMask</i>	New file creation mask

Returns

Previous value of the mask.

9.1.6.251 hpss_Undelete()

```
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	<i>Path</i>	Path of file to undelete
in	<i>Flag</i>	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.

9.1.6.252 hpss_UndeleteHandle()

```
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	<i>ObjHandle</i>	Parent object handle
in	<i>Path</i>	Path of file to undelete
in	<i>Ucred</i>	User credentials
in	<i>Flag</i>	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.

9.1.6.253 hpss_Unlink()

```
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	<i>Path</i>	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 be overwritten by the rename, and be moved to the trashcan corresponding to the deleting user and the Path's location.

9.1.6.254 hpss_UnlinkHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file to get statistics
in	<i>Ucred</i>	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 be overwritten by the rename, and be moved to the trashcan corresponding to the deleting user and the Path's location.

9.1.6.255 hpss_UnlinkHandleImmediate()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file to get statistics
in	<i>Ucred</i>	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.

Return values

<i>-EPERM</i>	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.

9.1.6.256 hpss_UnlinkImmediate()

```
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	<i>Path</i>	Path of file to unlink

Return values

<i>0</i>	Unlink was successful.
<i>-EACCES</i>	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.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	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.

9.1.6.257 hpss_UpdateACL()

```
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	<i>Path</i>	path of file
in	<i>Options</i>	processing options
in	<i>ACL</i>	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.

9.1.6.258 hpss_UpdateACLHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file
in	<i>Ucred</i>	user credentials
in	<i>Options</i>	processing options
in	<i>ACL</i>	ACL array to be updated

Return values

<i>0</i>	No error.
<i>-EACCES</i>	Search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path or ACL parameter is a NULL pointer.
<i>-EINVAL</i>	There are two sets of mutually exclusive flags available through the Options paramter. An invalid combinations of flags were provided.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.

9.1.6.259 hpss_UserAttrCloseCursor()

```
int hpss_UserAttrCloseCursor (
    int SubsystemId,
    const hpss_cursor_id_t * CursorId )
```

Close a cursor.

Parameters

in	<i>SubsystemId</i>	Subsystem Id
in	<i>CursorId</i>	CursorId

Close the given cursor. This will release the core server resources allocated for this cursor and finish the transaction.

Return values

<i>-EPERM</i>	The user who requested to close a cursor does not match the user who initiated the operation.
<i>-EFAULT</i>	The CursorId parameter is NULL.
<i>-ESRCH</i>	The cursor requested to be closed could not be found and may have already been closed due to a timeout.

9.1.6.260 hpss_UserAttrDeleteAttrHandle()

```
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	<i>Obj</i>	Parent Object
in	<i>Path</i>	Namespace Path
in	<i>Ucred</i>	user credentials
in	<i>Attrs</i>	Attributes
in	<i>Schema</i>	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

<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	One of the attribute pairs in Attr is invalid.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	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.

9.1.6.261 **hpss_UserAttrDeleteAttrs()**

```
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	<i>Path</i>	Namespace Path
in	<i>Attrs</i>	Attributes
in	<i>Schema</i>	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

<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.

Return values

<i>-EINVAL</i>	One of the attribute pairs in Attr is invalid.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	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.

9.1.6.262 **hpss_UserAttrGetAttrHandle()**

```
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	<i>Obj</i>	Parent Object
in	<i>Path</i>	Namespace Path
in	<i>Ucred</i>	user credentials
out	<i>Attrs</i>	Attributes and Values
in	<i>XMLFlag</i>	XML Flag
in	<i>XMLSize</i>	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

<i>-EACCES</i>	The user does not have permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer and Obj is NULL.
<i>-EINVAL</i>	One of the attribute pairs in Attr is invalid.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named object does not exist, or the Path argument points to an empty string.
<i>-ENOATTR</i>	No user attributes exist for the specified object.
<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.

Return values

<i>-ERANGE</i>	The length of the data was too large to be returned completely.
<i>-EDOM</i>	The XMLSize was too large.

9.1.6.263 **hpss_UserAttrGetAttrs()**

```
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	<i>Path</i>	Namespace Path
out	<i>Attrs</i>	Attributes and Values
in	<i>XMLFlag</i>	XML Flag
in	<i>XMLSize</i>	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

<i>-EACCES</i>	The user does not have read permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	One of the attribute pairs in Attr is invalid, or an invalid XMLFlag was used.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named object does not exist, or the Path argument points to an empty string.
<i>-ENOATTR</i>	No user attributes exist for the specified object.
<i>-ERANGE</i>	The length of the data was too large to be returned completely.
<i>-EDOM</i>	The XMLSize was too large.

9.1.6.264 **hpss_UserAttrListAttrHandle()**

```
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	<i>Obj</i>	Parent Object
in	<i>Path</i>	Path
in	<i>Ucred</i>	user credentials
out	<i>Attrs</i>	Attributes
in	<i>Flags</i>	Flags
in	<i>XMLSize</i>	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.
-ENOATTR	No user attributes exist for the specified object.
-EACCES	The user does not have write permission on the namespace 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.

9.1.6.265 hpss_UserAttrListAttrs()

```

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	<i>Path</i>	Namespace Path
out	<i>Attrs</i>	Attributes
in	<i>Flags</i>	Flags
in	<i>XMLSize</i>	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.
-ENOATTR	No user attributes exist for the specified object.
-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.

9.1.6.266 hpss_UserAttrReadObj()

```
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	<i>SubsystemId</i>	Subsystem Id
out	<i>ObjectList</i>	Objects
in	<i>CursorId</i>	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.

9.1.6.267 hpss_UserAttrReadXML()

```
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	<i>SubsystemId</i>	Subsystem Id
out	<i>XML</i>	XML Returned
in	<i>CursorId</i>	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.

9.1.6.268 hpss_UserAttrReadXMLObj()

```
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	<i>SubsystemId</i>	Subsystem Id
out	<i>ObjectList</i>	Objects
out	<i>XML</i>	XML Returned
in	<i>CursorId</i>	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

<i>-EACCES</i>	The user requested a read from this cursor does not match the user who initiated the operation.
<i>-EFAULT</i>	The ObjectList, XML, or CursorId parameters are NULL.
<i>-ENOENT</i>	There were no entries left to read.
<i>-ESRCH</i>	The cursor requested for reading was not found; it was likely invalidated due to prolonged inactivity.
<i>-ERANGE</i>	One or more of the results contained more data than could be retrieved.

9.1.6.269 hpss_UserAttrSetAttrHandle()

```
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	<i>Obj</i>	Parent Object
in	<i>Path</i>	Namespace Path
in	<i>Ucred</i>	user credentials
in	<i>Attrs</i>	Attributes
in	<i>Schema</i>	Schema

Return values

<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer and Obj is NULL.
<i>-EINVAL</i>	One of the attribute pairs in Attr is invalid.

Return values

<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	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.

9.1.6.270 **hpss_UserAttrSetAttrs()**

```
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	<i>Path</i>	Namespace Path
in	<i>Attrs</i>	Attributes
in	<i>Schema</i>	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

<i>0</i>	Success
<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path or Attrs parameter is a NULL pointer.
<i>-EINVAL</i>	One of the attribute pairs in Attr is invalid.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named object does not exist, or the Path argument points to an empty string.

9.1.6.271 hpss_UserAttrXQueryGet()

```
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	<i>Path</i>	Namespace Path
in	<i>XQuery</i>	XQuery
in, out	<i>Buffer</i>	Buffer
in	<i>XMLSize</i>	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.
-ENOATTR	No user attributes exist for the specified object.
-ERANGE	The requested data was too large to be returned completely.

9.1.6.272 hpss_UserAttrXQueryGetHandle()

```
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	<i>Obj</i>	Parent Object
in	<i>Path</i>	Namespace Path
in	<i>Ucred</i>	user credentials
in	<i>XQuery</i>	XQuery
in, out	<i>Buffer</i>	Buffer
	<i>XMLSize</i>	XML Size

Return values

<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	Invalid XQuery.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named object does not exist, or the Path argument points to an empty string.
<i>-ENOATTR</i>	No user attributes exist for the specified object.
<i>-ERANGE</i>	The requested data was too large to be returned completely.

9.1.6.273 **hpss_UserAttrXQueryUpdate()**

```
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	<i>Path</i>	Namespace Path
in	<i>XQuery</i>	XQuery string
in	<i>Schema</i>	Schema

Return values

<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	Invalid XQuery.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	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.

9.1.6.274 **hpss_UserAttrXQueryUpdateHandle()**

```
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	<i>Obj</i>	Namespace Object
in	<i>Path</i>	Namespace Path
in	<i>Ucred</i>	user credentials
in	<i>XQuery</i>	XQuery
in	<i>Schema</i>	Schema

Return values

<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	Invalid XQuery.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	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.

9.1.6.275 hpss_Utime()

```

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	<i>Path</i>	path of the object
in	<i>Times</i>	new time values

Return values

0	Success

Return values

<i>-EACCES</i>	Search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.

9.1.6.276 **hpss_Write()**

```
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	<i>Fildes</i>	ID of object to be written
in	<i>Buf</i>	Buffer from which to send data
in	<i>Nbyte</i>	number of bytes to write

Return values

0	End of File reached (EOF)
>0	Number of bytes successfully written
<i>-EBADF</i>	The specified file descriptor does not correspond to a file opened for writing.
<i>-EBUSY</i>	The file is currently in use by another client thread.
<i>-EFAULT</i>	The Buf parameter is out of range.
<i>-EFBIG</i>	The write operation would cause the file to exceed the system-imposed maximum file length.
<i>-EIO</i>	An input/output or HPSS internal error occurred.
<i>-ENOSPC</i>	There is no free space remaining to satisfy the write request.

9.1.6.277 **hpss_WriteList()**

```
int hpss_WriteList (
    const hpss_IOD_t * IODPtr,
```



```
uint32_t Flags,
hpss_IOR_t * IORPtr )
```

Issue a write IOD to HPSS.

Parameters

in	<i>IODPtr</i>	IO description
in	<i>Flags</i>	IO specific flags
out	<i>IORPtr</i>	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.

9.2 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.

9.2.1 Detailed Description

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

9.2.2 Function Documentation

9.2.2.1 Common_hpss_UnlinkHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file to get statistics
in	<i>Ucred</i>	user credentials
in	<i>Immediate</i>	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.

9.2.2.2 hpss_Close()

```
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	<i>Fildes</i>	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](#)

9.2.2.3 hpss_Creat()

```
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 *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	<i>Path</i>	Path to file to be opened
in	<i>Mode</i>	Desired file perms if create
in	<i>HintsIn</i>	Desired class of service
in	<i>HintsPri</i>	Priorities of hint struct
out	<i>HintsOut</i>	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.

9.2.2.4 hpss_Create()

```
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 'Path', with permissions as specified by 'Mode' and using the class of service values specified by 'HintsIn' and 'HintsPri', if non-NULL.

Parameters

in	<i>Path</i>	Path to file to be opened
in	<i>Mode</i>	Desired file perms if create
in	<i>HintsIn</i>	Desired class of service
in	<i>HintsPri</i>	Priorities of hint struct
out	<i>HintsOut</i>	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.

9.2.2.5 hpss_CreateHandle()

```
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 *Path*, with permissions as specified by *Mode* and using the class of service values specified by *HintsIn* and *HintsPri*, if non-NULL.

Parameters

in	<i>ObjHandle</i>	Parent object handle
in	<i>Path</i>	Path to file to be opened
in	<i>Mode</i>	Desired file perms if create
in, out	<i>Ucred</i>	User credentials
in	<i>HintsIn</i>	Desired class of service
in	<i>HintsPri</i>	Priorities of hint struct
out	<i>HintsOut</i>	Granted class of service
out	<i>AttrsOut</i>	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.

9.2.2.6 hpss_Open()

```
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	<i>Path</i>	Names the file to be opened or created.
in	<i>Oflag</i>	Specifies the file status and access modes to be assigned. Applicable values give 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	<i>Mode</i>	Specifies the file mode for a file that is created as a result of O_CREAT.
in	<i>HintsIn</i>	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.

Parameters

in	<i>HintsPri</i>	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	<i>HintsOut</i>	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.

9.2.2.7 hpss_OpenBitfile()

```
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	<i>BitfileObj</i>	Bitfile Object Handle (usually obtained using <code>hpss_FileGetAttributes</code>)
in	<i>Oflag</i>	Specifies file status and file access modes to be assigned. Applicable values below may be OR'd together. • <code>O_RDONLY</code> • <code>O_WRONLY</code> • <code>O_RDWR</code> • <code>O_APPEND</code> • <code>O_TRUNC</code> • <code>HPSS_O_NO_TAPE</code> • <code>HPSS_O_EXCL</code> • <code>HPSS_O_STAGE_ASYNC</code> • <code>HPSS_O_STAGE_BKGRD</code>
in	<i>Ucred</i>	User Credentials

Return values

<code>-EACCESS</code>	The client does not have permission for the requested file access.
<code>-EFAULT</code>	The <code>BitFileId</code> parameter is a NULL pointer.
<code>-EINPROGRESS</code>	The file is currently being staged. The open should be retried at a later time.
<code>-EINVAL</code>	<code>Oflag</code> is not valid.
<code>-EMFILE</code>	The client file table is already full.
<code>-ENFILE</code>	Too many bitfiles are already open in the system.
<code>-ENOENT</code>	No entry exists for the specified bitfile id.
<code>-EREMOTE</code>	The caller requested the <code>HPSS_O_NO_TAPE</code> 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.

9.2.2.8 hpss_OpenBitfileVAttr()

```
int hpss_OpenBitfileVAttr (
    const hpss_vattr_t * FileAttrs,
    int Oflag,
    const sec_cred_t * Ucred,
    hpss_cos_hints_t * HintsOut,
    uint64_t * SegmentSize )
```


Open a bitfile using `vattr`.

The 'hpss_OpenBitfileVAttr' function is a no-frills interface for opening 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.

9.2.2.9 hpss_OpenHandle()

```
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	<i>ObjHandle</i>	Parent object handle; the parent directory.
in	<i>Path</i>	Names the file to be opened or created.
in	<i>Oflag</i>	Specifies the file status and access modes to be assigned. Applicable values give 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	<i>Ucred</i>	User Credentials
in	<i>Mode</i>	Specifies the file mode for a file that is created as a result of O_CREAT.
in	<i>Ucred</i>	User credentials
in	<i>HintsIn</i>	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	<i>HintsPri</i>	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	<i>HintsOut</i>	The hints which were used for this file open; note that these might differ from the hints provided.
out	<i>AttrsOut</i>	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

<i>-EACCESS</i>	One of the following occurred: • Search permission is denied on a component of the path prefix. • The file exists and the permissions specified by <i>Oflag</i> are denied. • The file does not exist and write permission is denied for the parent directory of the file to be created. • <i>O_TRUNC</i> is specified and write permission is denied.
<i>-EEXIST</i>	<i>O_CREAT</i> and <i>O_EXCL</i> are set and the named file exists.
<i>-EFAULT</i>	The <i>Path</i> parameter is a NULL pointer.
<i>-EINPROGRESS</i>	The file is currently being staged. The open should be retried at a later time.
<i>-EINVAL</i>	<i>Oflag</i> is not valid, or one or more values input in <i>HintsIn</i> are invalid.
<i>-EISDIR</i>	The named file is a directory. Note that opening directories via this function is not supported in any mode.
<i>-EMFILE</i>	The client open file table is already full.
<i>-ENFILE</i>	Too many open files on the system.
<i>-ENAMETOOLONG</i>	The length of the <i>Path</i> string, or a path name component exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist and the <i>O_CREAT</i> flag was not specified, or the <i>Path</i> argument points to an empty string.
<i>-ENOSPC</i>	Resources could not be allocated for the new file.
<i>-ENOTDIR</i>	A component of the path is not a directory.
<i>-EREMOTE</i>	The caller requested the <i>HPSS_O_NO_TAPE</i> 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.

9.2.2.10 hpss_ReopenBitfile()

```
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	<i>Fildes</i>	Descriptor to be reopened
in	<i>BitfileObj</i>	Bitfile Object to Open

Parameters

in	<i>Oflag</i>	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	<i>Ucred</i>	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	File descriptor 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.

9.2.2.11 hpss_Unlink()

```
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	<i>Path</i>	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 be overwritten by the rename, and be moved to the trashcan corresponding to the deleting user and the Path's location.

9.2.2.12 hpss_UnlinkHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file to get statistics
in	<i>Ucred</i>	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.

Return values

<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	The file named by Path is a directory.

Note

If trashcans are enabled, Path will be overwritten by the rename, and be moved to the trashcan corresponding to the deleting user and the Path's location.

9.2.2.13 hpss_UnlinkHandleImmediate()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file to get statistics
in	<i>Ucred</i>	user credentials

Return values

<i>0</i>	Unlink was successful
<i>-EACCES</i>	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.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	ObjHandle is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	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.

9.2.2.14 hpss_UnlinkImmediate()

```
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.

9.3 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_PreferedReadOrder](#) (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.
- 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_SetNextIORquestID](#) (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_GetUniqueRequestID 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.

9.3.1 Detailed Description

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

9.3.2 Function Documentation

9.3.2.1 hpss_CopyFile()

```
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	<i>SrcFildes</i>	open desc for src file
in	<i>DestFildes</i>	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.

9.3.2.2 hpss_Fclear()

```
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	<i>Fildes</i>	Specifies the file descriptor identifying the open file
in	<i>Length</i>	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](#)

9.3.2.3 hpss_FclearOffset()

```
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	<i>Fildes</i>	Specifies the file descriptor identifying the open file
in	<i>Offset</i>	Specifies where to begin clearing the file
in	<i>Length</i>	Specifies the number of bytes to be cleared

Return values

0	Success
-EINVAL	The <i>Offset</i> or <i>Length</i> 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](#)

9.3.2.4 hpss_Fpreallocate()

```
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	<i>Fildes</i>	ID of open object
in	<i>Length</i>	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.

9.3.2.5 hpss_Ftruncate()

```
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	<i>Fildes</i>	ID of open object
in	<i>Length</i>	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.

9.3.2.6 hpss_Lseek()

```
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	<i>Fildes</i>	ID of open object
in	<i>Offset</i>	# of bytes to calculate new offset
in	<i>Whence</i>	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.

9.3.2.7 hpss_PREFERREDReadOrder()

```
signed32 hpss_PREFERREDReadOrder (
    uint32_t SubsysId,
    hpss_uid_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	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read context to use
in	<i>UserCredsP</i>	User Credentials
in	<i>ItemsToCheck</i>	List of read requests to check on (or NULL)
in	<i>Flags</i>	Flags • <code>CORE_RQ_ONLY_NEW_READY</code>. Only return new, ready read requests.
in	<i>Delay</i>	How long to check for requests which are ready to run
out	<i>ItemsReady</i>	Which items are ready to run
out	<i>ItemsError</i>	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

<i>0</i>	Success
<i>-EFAULT</i>	One or more of <i>ContextP</i> , <i>ItemsReady</i> , or <i>ItemsError</i> is NULL
<i>-ENOENT</i>	<i>ItemsToCheck</i> has no entries, or nothing is ready
<i>Other</i>	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.

9.3.2.8 hpss_Read()

```
ssize_t hpss_Read (
    int Fildes,
    void * Buf,
    size_t Nbyte )
```

Reads a number of bytes from a file.

Parameters

in	<i>Fildes</i>	ID of object to be read
in	<i>Buf</i>	Buffer in which to receive data
in	<i>Nbyte</i>	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

<i>positive</i>	Number of Bytes Read
<i>0</i>	End of file reached
<i>-EBADF</i>	The specified file descriptor does not correspond to a file opened for reading.
<i>-EBUSY</i>	The file is currently in use by another client thread.
<i>-EFAULT</i>	The Buf parameter is NULL.
<i>-EIO</i>	An input/output or HPSS internal error occurred.

9.3.2.9 hpss_ReadItem()

```
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	<i>Fildes</i>	ID of object to be read
in	<i>Buf</i>	Buffer in which to receive data
in	<i>Nbyte</i>	number of bytes to read
in	<i>Item</i>	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

<i>positive</i>	Number of Bytes Read
<i>0</i>	End of file reached
<i>-EBADF</i>	The specified file descriptor does not correspond to a file opened for reading.
<i>-EBUSY</i>	The file is currently in use by another client thread.
<i>-EFAULT</i>	The Buf or Item parameter is NULL.
<i>-EIO</i>	An input/output or HPSS internal error occurred.
<i>-EIO</i>	Failed to set the request ID
<i>Other</i>	POSIX error indicating the nature of the error

9.3.2.10 hpss_ReadQueueAdd()

```
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	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read context to add to
in	<i>Item</i>	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

9.3.2.11 hpss_ReadQueueAddMany()

```
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	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read context to add to
in	<i>ItemList</i>	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

9.3.2.12 hpss_ReadQueueCreateContext()

```
signed32 hpss_ReadQueueCreateContext (
    uint32_t SubsysId,
    sec_cred_t * UserCredsP,
    hpss_uuid_t * ContextP )
```

Create a new read queue context.

Parameters

in	<i>SubsysId</i>	Subsystem ID
out	<i>ContextP</i>	Context UUID

Return values

0	Success
-EFAULT	ContextP is NULL
Other	POSIX error indicating the nature of the error

9.3.2.13 hpss_ReadQueueList()

```
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	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read context to use
in	<i>UserCredsP</i>	User Credentials
in	<i>VVID</i>	VV to List (optional)
in	<i>Start</i>	Initial list index to start listing from in ContextP
in	<i>Max</i>	Maximum number of items to return
out	<i>List</i>	Sorted list of entries in the VVIDs read queue

Return values

<i>0</i>	Success
<i>-EFAULT</i>	ContextP or List is NULL
<i>Other</i>	POSIX error indicating the nature of the error

9.3.2.14 **hpss_ReadQueueRemove()**

```
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	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read context to use
in	<i>UserCredsP</i>	User Credentials
in	<i>RequestList</i>	List of requests to remove from the queue

Return values

<i>0</i>	Success
<i>-EFAULT</i>	Context or RequestList is NULL
<i>Other</i>	POSIX error indicating the nature of the error

9.3.2.15 **hpss_ReadQueueRemoveContext()**

```
signed32 hpss_ReadQueueRemoveContext (
    uint32_t SubsysId,
    hpss_uuid_t * ContextP,
    sec_cred_t * UserCredsP,
    int Flags )
```

Remove a read queue context.

Parameters

in	<i>SubsysId</i>	Subsystem ID
in	<i>ContextP</i>	Read queue context to use
in	<i>Flags</i>	Read Context Flags • HPSS_REMOVE_CONTEXT_ABORT - Abort all existing requests

Return values

<i>0</i>	Success
<i>-EFAULT</i>	ContextP is NULL
<i>Other</i>	POSIX error indicating the nature of the error

9.3.2.16 hpss_SetFileOffset()

```
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 offsets (ala lseek()).

Parameters

in	<i>Fildes</i>	ID of open object
in	<i>OffsetIn</i>	# of bytes to calculate new offset
in	<i>Whence</i>	Origin for the seek
in	<i>Direction</i>	OffsetIn is either forward or backward
out	<i>OffsetOut</i>	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

<i>-EBADF</i>	The specified file descriptor does not refer to an open file.
<i>-EBUSY</i>	The file is currently in use by another client thread.
<i>-EFAULT</i>	OffsetOut is a NULL pointer.
<i>-EINVAL</i>	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.
<i>-ENOSPC</i>	Resources could not be allocated to satisfy the request.

9.3.2.17 hpss_SetNextIORequestID()

```
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_↵GetUniqueRequestID` or from `hpss_ReadQueueAdd`.

Sets the next request id

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>EFAULT</i>	NULL Request ID provided
<i>EINVAL</i>	Invalid request ID provided

Note

This should not be called except with a request ID provided via the `hpss_ReadQueueAddItem` or `hpss_↵ReadQueueAddMany` APIs.

9.3.2.18 `hpss_Truncate()`

```
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	<i>Path</i>	Pathname of file
in	<i>Length</i>	New length

Return values

<i>0</i>	Success
<i>-EACCES</i>	Search permission is denied on a component of the path prefix, or write permission is denied on the file.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	Path specifies a directory.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The specified path name does not exist.
<i>-ENOTDIR</i>	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.

9.3.2.19 hpss_TruncateHandle()

```
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	<i>ObjHandle</i>	Handle to file
in	<i>Path</i>	file name
in	<i>Length</i>	New length
in	<i>Ucred</i>	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.

9.3.2.20 hpss_Write()

```
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	<i>Fildes</i>	ID of object to be written
in	<i>Buf</i>	Buffer from which to send data
in	<i>Nbyte</i>	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.

9.4 Fileset and Junction

Functions which operate upon filesets and junctions.

Functions

- int [hpss_FilesetCreate](#) (const hpss_svr_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 *FilesetHandle)
Deletes an existing fileset.
- int [hpss_FilesetGetAttributes](#) (const char *FilesetName, const uint64_t *FilesetId, const ns_ObjHandle_t *FilesetHandle, const hpss_svr_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.

- 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_ObjHandle_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.

9.4.1 Detailed Description

Functions which operate upon filesets and junctions.

9.4.2 Function Documentation

9.4.2.1 hpss_FilesetCreate()

```
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	<i>CoreServerID</i>	Core Server ID
in	<i>CreateOptions</i>	creation options CORE_NS_FILESET - Create Core Server fileset metadata
in	<i>FilesetAttrBits</i>	Fileset attributes to set
in	<i>FilesetAttrs</i>	Fileset attributes
in	<i>ObjectAttrBits</i>	Object attributes to set
in	<i>ObjectAttrs</i>	Object attributes
out	<i>RetFilesetAttrBits</i>	Fileset attr to return
out	<i>RetObjectAttrBits</i>	Dir attr to return
out	<i>RetFilesetAttrs</i>	Returned fileset attr
out	<i>RetObjectAttrs</i>	Returned dir attr
out	<i>FilesetHandle</i>	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.

9.4.2.2 hpss_FilesetDelete()

```
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	<i>FilesetName</i>	Fileset name
in	<i>FilesetId</i>	Fileset ID
in	<i>FilesetHandle</i>	Fileset object handle

Return values

0	Success

Return values

<i>-EACCES</i>	The user is not the root user or a trusted user.
<i>-EFAULT</i>	The Name, FilesetID and FilesetHandle arguments are all NULL pointers.
<i>-EINVAL</i>	More that one type of fileset identifier was specified.
<i>-ENOENT</i>	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.

9.4.2.3 hpss_FilesetGetAttributes()

```
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	<i>FilesetName</i>	Fileset name
in	<i>FilesetId</i>	Fileset ID
in	<i>FilesetHandle</i>	Fileset object handle
in	<i>CoreServerID</i>	Core server ID for fileset
in	<i>FilesetAttrBits</i>	Fileset attribute bits
out	<i>FilesetAttrs</i>	Returned attributes

Return values

<i>-EACCES</i>	The user is not the root user or a trusted user.
<i>-EFAULT</i>	The FilesetName, FilesetID and FilesetHandle arguments are all NULL pointers.
<i>-EINVAL</i>	More that one type of fileset identifier was specified or invalid file set attribute bits were specified.
<i>-ENOENT</i>	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.

9.4.2.4 hpss_FilesetListAll()

```
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	<i>OffsetIn</i>	offset to start at
in	<i>Entries</i>	number of entries to return
out	<i>End</i>	hit end of the list
out	<i>OffsetOut</i>	offset of last entry
out	<i>FSentPtr</i>	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.

9.4.2.5 hpss_FilesetSetAttributes()

```
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	<i>FilesetName</i>	Fileset name
in	<i>FilesetId</i>	Fileset ID
in	<i>FilesetHandle</i>	Fileset object handle
in	<i>FilesetAttrBitsIn</i>	Fileset attribute bits
in	<i>FilesetAttrsIn</i>	Fileset attributes
in	<i>FilesetAttrBitsOut</i>	Returned set attribute bits
out	<i>FilesetAttrsOut</i>	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 than 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.

9.4.2.6 hpss_GetJunctionAttrs()

```
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	<i>Path</i>	path of file to get statistics
out	<i>AttrOut</i>	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.

Return values

<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.

9.4.2.7 hpss_GetJunctions()

```
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	<i>SubsystemID</i>	Subsystem identifier
in	<i>OffsetIn</i>	offset to start at
in	<i>Entries</i>	number of entries to return
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	offset of last entry
out	<i>JentPtr</i>	junction entry information

Return values

0	Success
<i>-EFAULT</i>	The End, OffsetOut or JentPtr parameter is a NULL pointer.
<i>-EINVAL</i>	The Entries parameter specifies zero entries to read.

9.4.2.8 hpss_JunctionCreate()

```
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	<i>Path</i>	Path of the new junction
in	<i>SourceHandle</i>	Directory or fileset handle
out	<i>JunctionHandle</i>	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.

9.4.2.9 hpss_JunctionCreateHandle()

```
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	<i>ParentHandle</i>	directory to insert junction
in	<i>Path</i>	path name of the new junction
in	<i>SourceHandle</i>	Directory or fileset handle
in	<i>Ucred</i>	user credentials
out	<i>JunctionHandle</i>	Junction handle

Returns

The object handle for the new junction

Return values

<i>0</i>	Success
<i>-EFAULT</i>	Either the Path, SourceHandle, or JunctionHandle parameter is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system imposed limit, or a component of the pathname exceeds the system imposed limit.
<i>-ENOENT</i>	The Path argument points to an empty string.
<i>-EEXIST</i>	The named path already exists in the HPSS name space.
<i>-EACCES</i>	The requesting client is not the root user or a trusted user with write permissions.
<i>-EINVAL</i>	The SourceHandle parameter doesn't point to a directory handle or the Path is invalid.

9.4.2.10 `hpss_JunctionDelete()`

```
int hpss_JunctionDelete (
    const char * Path )
```

Deletes a junction.

The 'hpss_JunctionDelete' function is called to delete a junction.

Parameters

<i>in</i>	<i>Path</i>	Path of junction to delete

Return values

<i>0</i>	Success
<i>-ENOENT</i>	The Path parameter is an empty string or doesn't refer to an existing object.
<i>-EFAULT</i>	The Path parameter is NULL.
<i>-EINVAL</i>	The Path parameter doesn't refer to a junction.
<i>-EACCES</i>	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.

9.4.2.11 `hpss_JunctionDeleteHandle()`

```
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	<i>ParentHandle</i>	Parent directory
in	<i>Path</i>	Name of junction
in	<i>Ucred</i>	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.

9.5 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 SelfFlags, 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 SelfFlags, 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 SelfFlags, 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 a 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, [purge_lock_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 SelfFlagsIn, const hpss_vattr_t *AttrIn, uint64_t *SelfFlagsOut, 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.

9.5.1 Detailed Description

Functions which operate upon HPSS file attributes.

9.5.2 Function Documentation

9.5.2.1 `hpss_Access()`

```
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	<i>Path</i>	Path of file to check access rights
in	<i>Amode</i>	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 <i>Amode</i> are denied, or search permission is denied on a component path of the path prefix.
-EFAULT	The <i>Path</i> parameter is a NULL pointer.
-EINVAL	An invalid value was specified for <i>Amode</i> .
-ENAMETOOLONG	The length of the <i>Path</i> 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 <i>Path</i> argument points to an empty string.
-ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

See also

[hpss_AccessHandle](#), [hpss_Chmod](#), [hpss_FileSetAttributes](#)

9.5.2.2 `hpss_AccessHandle()`

```
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_AccessHandle` 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	<i>ObjHandle</i>	Handle to the parent object
in	<i>Path</i>	Path name of the file being checked
in	<i>Amode</i>	Indicates the type of file access being checked. Refer to POSIX.1 for possible values.
in	<i>Ucred</i>	User credentials

Return values

0	No error, caller has access.
-EACCES	The permissions specified by <i>Amode</i> are denied, or search permission is denied on a component of the path prefix.
-EINVAL	An invalid value was specified for <i>Amode</i> or the <i>Path</i> parameter is a NULL pointer.
-ENAMETOOLONG	The length of the <i>Path</i> 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 <i>Path</i> argument points to an empty string.
-ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

See also

[hpss_Access](#), [hpss_Chmod](#), [hpss_FileSetAttributes](#)

9.5.2.3 hpss_Chacct()

```
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	<i>Path</i>	Name of the file whose account code will be modified
in	<i>AcctCode</i>	New accounting code for the file at <i>Path</i>

Return values

-EACCES	Search permission is denied on a component of the <i>Path</i> prefix.
-EFAULT	<i>Path</i> is a NULL pointer.
-ENAMETOOLONG	The length of the <i>Path</i> 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 <i>Path</i> argument points to an empty string.
-ENOTDIR	A component of the <i>Path</i> 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.

Return values

0	No error. Account code altered as directed.

See also

[hpss_ChacctByName](#)

9.5.2.4 hpss_ChacctByName()

```
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	<i>Path</i>	File or directory for which the account code is being changed
in	<i>AcctName</i>	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.

9.5.2.5 hpss_Chmod()

```
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	<i>Path</i>	path to the object
in	<i>Mode</i>	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.

9.5.2.6 hpss_Chown()

```
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	<i>Path</i>	path to the object
in	<i>Owner</i>	desired new owner ID
in	<i>Group</i>	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.

9.5.2.7 hpss_FileGetAttributes()

```
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 referred to by *Path*.

Parameters

in	<i>Path</i>	Path to the object
out	<i>AttrOut</i>	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.

9.5.2.8 hpss_FileGetAttributesBitfile()

```
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 referred to by 'Bitfile'. It returns one of possibly multiple path names to the bitfile and the file attributes structure for the bitfile.

Parameters

in	<i>BitfileObj</i>	Bitfile Object
out	<i>Path</i>	Path to the object
out	<i>AttrOut</i>	Object Attributes

Return values

<i>0</i>	No error, caller has access.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The Path or AttrOut parameter is a NULL pointer.
<i>-EINVAL</i>	BitfileObj is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	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.

9.5.2.9 hpss_FileGetAttributesHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path to the object
in	<i>Ucred</i>	user credentials
out	<i>AttrOut</i>	attributes after query

Return values

<i>0</i>	No error, caller has access.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The Path or AttrOut parameter is a NULL pointer.
<i>-EINVAL</i>	ObjHandle is NULL.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

Return values

<i>-ENOENT</i>	The named file does not exist, or the <i>Path</i> argument points to an empty string.
<i>-ENOTDIR</i>	A component of the <i>Path</i> 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.

9.5.2.10 `hpss_FileGetXAttributes()`

```
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 referred 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	<i>Path</i>	Path to the object
in	<i>Flags</i>	Flags for storage attrs • <code>API_GET_STATS_FOR_LEVEL</code> - Returns bitfile attributes at the storage level specified by the <i>StorageLevel</i> argument • <code>API_GET_STATS_FOR_1STLEVEL</code> - Returns bitfile attributes at th first storage level whether or not it contains bitfile data. • <code>API_GET_STATS_FOR_OPTIMIZE</code> - Returns only <i>StripeWidth</i> and <i>OptimumAccessSize</i> for storage level zero • <code>API_GET_STATS_FOR_ALL_LEVELS</code> - Returns bitfile attributes across all storage levels • <code>API_GET_XATTRS_NO_BLOCK</code> - Causes the operation to be non-blocking, however this data may be inconsistent with metadata.
in	<i>StorageLevel</i>	Storage level to query
out	<i>AttrOut</i>	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.SCAAttrib[i].NumberOfVVs; j++)
    {
        if (AttrOut.SCAAttrib[i].VVAttrib[j].PVList != NULL)
        {
            free (AttrOut.SCAAttrib[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.

9.5.2.11 hpss_FileGetXAttributesHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path to the object
in	<i>Ucred</i>	user credentials
in	<i>Flags</i>	flags for storage attrs
in	<i>StorageLevel</i>	Storage level to query
out	<i>AttrOut</i>	attributes after query

Return values

<i>0</i>	No error, caller has access.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The Path or AttrOut parameter is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.

Warning

The storage level information must be freed by the application.

9.5.2.12 hpss_FileSetAttributes()

```
int hpss_FileSetAttributes (
    const char * Path,
    hpss_fileattrbits_t SelfFlags,
    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 referred to by 'Path'.

Parameters

in	<i>Path</i>	path to the object

Parameters

in	<i>SelfFlags</i>	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
Generated on Tue Sep 23 2025 for HPSS 11.3 Programmer's Reference		• CORE_ATTR_USER_PERMS • CORE_ATTR_WRITE_COUNT

Parameters

in	<i>AttrIn</i>	input attributes
out	<i>AttrOut</i>	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.

9.5.2.13 `hpss_FileSetAttributesBitfile()`

```
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 referred to by 'BitfileID'. It returns one of possibly multiple path names to the bitfile and the file attributes structure for the entry.

Parameters

in	<i>BitfileObj</i>	Bitfile Object
in	<i>SelFlags</i>	attributes fields to set; refer to <code>hpss_SetFileAttributes</code> for a list
in	<i>AttrIn</i>	input attributes
out	<i>AttrOut</i>	attributes after query
out	<i>Path</i>	path to the object

Return values

<i>0</i>	No error, caller has access.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The AttrOut parameter is a NULL pointer.
<i>-EINVAL</i>	An attribute value or selection flag is invalid, or BitfileObj is NULL.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOSPC</i>	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.

9.5.2.14 `hpss_FileSetAttributesHandle()`

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path to the object
in	<i>Ucred</i>	user credentials
in	<i>SelFlags</i>	attributes fields to set; refer to <code>hpss_SetFileAttributes</code> for a list
in	<i>AttrIn</i>	input attributes
out	<i>AttrOut</i>	attributes after change

Return values

<i>0</i>	No error, caller has access.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.

Return values

-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.

9.5.2.15 `hpss_FileSetCOS()`

```
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 referred 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	<i>Path</i>	path to the object
in	<i>COSId</i>	change COS to this value
in	<i>StreamId</i>	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.

Return values

<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOSPC</i>	Resources could not be allocated to satisfy the request.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EOPNOTSUPP</i>	The requested change is not supported.
<i>-EPERM</i>	The client does not have the appropriate privileges to change the file's attributes.

9.5.2.16 hpss_FileSetCOSHandle()

```
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 referred 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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path to the object
in	<i>Ucred</i>	user credentials
in	<i>COSId</i>	change COS to this value
in	<i>StreamId</i>	used this Core Server stream

Return values

0	No error, caller has access.
<i>-ENOMEM</i>	Not Enough Memory
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	An attribute value or selection flag is invalid.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOSPC</i>	Resources could not be allocated to satisfy the request.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EOPNOTSUPP</i>	The requested change is not supported.
<i>-EPERM</i>	The client does not have the appropriate privileges to change the file's attributes.

9.5.2.17 hpss_Fstat()

```
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	<i>Fildes</i>	ID of open file/directory
out	<i>Buf</i>	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_OpenBitfileVAttr 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.

9.5.2.18 hpss_GetAttrHandle()

```
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	<i>ObjHandle</i>	Parent object handle
in	<i>Path</i>	Path of file to get attributes
in	<i>Ucred</i>	User credentials
out	<i>HandleOut</i>	Returned object handle for <i>Path</i>
out	<i>AttrOut</i>	Returned attributes for <i>Path</i>

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.

9.5.2.19 hpss_GetFileNotrunc()

```
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	<i>Path</i>	Pathname of file
out	<i>NotruncFlag</i>	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.

9.5.2.20 hpss_GetListAttrs()

```
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	<i>Path</i>	path of file to get statistics
out	<i>AttrOut</i>	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.

9.5.2.21 hpss_GetRawAttrHandle()

```
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	<i>ObjHandle</i>	Parent object handle
in	<i>Path</i>	Path of file to get attributes
in	<i>Ucred</i>	User credentials
out	<i>HandleOut</i>	Returned object handle for <i>Path</i>
out	<i>AttrOut</i>	Returned attributes for <i>Path</i>

Return values

<i>0</i>	No error. Valid information returned.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The Path or AttrOut parameter is a NULL pointer.
<i>-EINVAL</i>	The ObjHandle parameter is NULL.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.

9.5.2.22 hpss_Lstat()

```
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	<i>Path</i>	path of file to get statistics
out	<i>Buf</i>	Returned statistics

Return values

<i>0</i>	No error. Statistics in buffer.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The Path or Buf parameter is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.

9.5.2.23 hpss_PurgeLock()

```
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	<i>Fildes</i>	ID of open object
in	<i>Flag</i>	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.

9.5.2.24 hpss_PurgeOnMigrate()

```
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	<i>Fildes</i>	Open file handle
in	<i>Flag</i>	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.

Return values

<i>-ESTALE</i>	The connection for this entry is not valid.
<i>-EINVAL</i>	An incorrect value was specified for Flag.

9.5.2.25 **hpss_SetAttrHandle()**

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file to set attributes
in	<i>Ucred</i>	user credentials
in	<i>SelFlagsIn</i>	indication of attrs to be set
in	<i>AttrIn</i>	new attribute values
out	<i>SelFlagsOut</i>	indication of attrs set
out	<i>AttrOut</i>	returned attributes

Return values

0	No error.
<i>-EACCES</i>	Search permission is denied for a component of the path prefix.
<i>-EFAULT</i>	The Path, AttrIn or AttrOut parameter is a NULL pointer.
<i>-EINVAL</i>	An attribute value or selection flag is invalid, or the ObjHandle is NULL.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOSPC</i>	Resources could not be allocated to satisfy the request.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EOPNOTSUPP</i>	The requested change is not supported.
<i>-EPERM</i>	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.

9.5.2.26 hpss_SetCOSByHints()

```
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	<i>Fildes</i>	ID of object to be read
in	<i>Flags</i>	operation flags to use
in	<i>HintsPtr</i>	COS hints desired
in	<i>PrioPtr</i>	Priorities of COS hints
out	<i>COSPtr</i>	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.

9.5.2.27 hpss_SetFileNotrunc()

```
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	<i>Path</i>	Pathname of file
in	<i>Flag</i>	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.

9.5.2.28 hpss_Stat()

```
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	<i>Path</i>	path of file to get statistics
out	<i>Buf</i>	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.

9.5.2.29 hpss_Utime()

```
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	<i>Path</i>	path of the object
in	<i>Times</i>	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.

9.6 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.

9.6.1 Detailed Description

Functions used to rename or remove names associated with files.

9.6.2 Function Documentation

9.6.2.1 hpss_Link()

```
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	<i>Existing</i>	Existing name of the object
in	<i>New</i>	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.

9.6.2.2 hpss_LinkHandle()

```
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	<i>ObjHandle</i>	handle of existing obj
in	<i>DirHandle</i>	handle of parent directory
in	<i>New</i>	New name of the object
in	<i>Ucred</i>	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.

9.6.2.3 hpss_LinkHandleParent()

```
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	<i>SrcDirHandle</i>	Handle to source parent directory
in	<i>SrcFile</i>	Path within source parent directory
in	<i>DestDirHandle</i>	Handle to destination parent directory
in	<i>DestFile</i>	New link path within destination parent directory
in	<i>Ucred</i>	user credentials

Return values

0	Link was successful.

Return values

<i>-EACCES</i>	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the new link.
<i>-EEXIST</i>	The object identified by DestObjHandle and DestPath already exists.
<i>-EFAULT</i>	The SrcDirHandle, DestDirHandle, SrcFile, or DestFile parameter is a NULL pointer.
<i>-EPERM</i>	The object specified by SrcDirHandle and SrcFile is a directory.
<i>-EMLINK</i>	The number of links to the source object would exceed the system-imposed limit.
<i>-ENAMETOOLONG</i>	The length of the DestFile argument exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
<i>-ENOENT</i>	New points to a null string.
<i>-ENOTDIR</i>	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.

9.6.2.4 hpss_Rename()

```
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	<i>Old</i>	Old name of the object
in	<i>New</i>	New name of the object

Return values

<i>0</i>	Rename was successful.
<i>-EACCES</i>	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.
<i>-EFAULT</i>	The Old or New parameter is a NULL pointer.
<i>-EISDIR</i>	The New argument points to a directory, and the Old argument points to a file that is not a directory.
<i>-EMLINK</i>	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.
<i>-ENAMETOOLONG</i>	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.
<i>-ENOENT</i>	The named file does not exist, or the Old or New argument points to an empty string.

Return values

<i>-ENOTDIR</i>	A component of the path prefix is not a directory.
<i>-ENOTEMPTY</i>	The path named by New is a directory containing entries other than dot and dotdot.

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 the deleting user and its location.

9.6.2.5 hpss_RenameHandle()

```
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	<i>OldHandle</i>	Old parent object handle
in	<i>OldPath</i>	Old name of the object
in	<i>NewHandle</i>	New parent object handle
in	<i>NewPath</i>	New name of the object
in	<i>Ucred</i>	User credentials

Return values

<i>0</i>	Rename was successful.
<i>-EACCES</i>	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.
<i>-EFAULT</i>	NewPath is a NULL pointer.
<i>-EINVAL</i>	The NewHandle or OldHandle parameter is NULL.
<i>-EISDIR</i>	The NewHandle argument points to a directory, and the OldHandle argument points to a file that is not a directory.
<i>-EMLINK</i>	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.
<i>-ENAMETOOLONG</i>	The length of OldPath or NewPath exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

Return values

<i>-ENOENT</i>	The named file does not exist, or the OldPath or NewPath argument points to an empty string.
<i>-ENOTDIR</i>	A component of the path prefix is not a directory.
<i>-ENOTEMPTY</i>	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 the deleting user and its location.

9.6.2.6 **hpss_Symlink()**

```
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	<i>Contents</i>	Desired contents of the new link
in	<i>Path</i>	Name of the link

Return values

<i>0</i>	Creation of symlink was successful.
<i>-EACCES</i>	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.
<i>-EFAULT</i>	The Path or Contents parameter is a NULL pointer.
<i>-EEXIST</i>	The specified link already exists.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	No entry exists for a component of the path name.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.

9.6.2.7 **hpss_SymlinkHandle()**

```
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	<i>ObjHandle</i>	handle of parent
in	<i>Contents</i>	Contents of the link
in	<i>Path</i>	New name of symlink
in	<i>Ucred</i>	user credentials
out	<i>AttrsOut</i>	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.

9.7 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)
Retrieves 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.

9.7.1 Detailed Description

Functions used to create, remove, and manipulate ACLs on files or directories.

9.7.2 Function Documentation

9.7.2.1 hpss_DeleteACL()

```
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	<i>Path</i>	Names the file for which the ACL is being removed.
in	<i>Options</i>	Bit vector used to specify what type of ACL is to be removed. One of: • - <i>HPSS_ACL_OPTION_OBJ</i> Normal ACL • - <i>HPSS_ACL_OPTION_IO</i> Initial Object ACL (only valid for directory objects) • - <i>HPSS_ACL_OPTION_IC</i> Initial Container ACL (only valid for directory objects)
in	<i>ACL</i>	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.

Return values

<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.

See also

[hpss_GetACL](#) [hpss_UpdateACL](#) [hpss_SetACL](#)

9.7.2.2 hpss_DeleteACLHandle()

```
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	<i>ObjHandle</i>	Parent object handle
in	<i>Path</i>	File for which the ACL is being removed
in	<i>Ucred</i>	User credentials
in	<i>Options</i>	Bit vector used to specify what type of ACL is to be removed. One of: • - <i>HPSS_ACL_OPTION_OBJ</i> Normal ACL • - <i>HPSS_ACL_OPTION_IO</i> Initial Object ACL (only valid for directory objects) • - <i>HPSS_ACL_OPTION_IC</i> Initial Container ACL (only valid for directory objects)
in	<i>ACL</i>	Points to the list of ACL entries to be removed.

Return values

0	No error.
<i>-EACCES</i>	Search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path or ACL parameter is a NULL pointer.
<i>-EINVAL</i>	Exactly one of the <i>HPSS_ACL_OPTION_*</i> bits must be set in the Options bit vector.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.
<i>-ESRCH</i>	A specified ACL entry did not match an existing ACL entry for the file.

See also

[hpss_GetACLHandle](#) [hpss_UpdateACLHandle](#) [hpss_SetACLHandle](#)

9.7.2.3 hpss_GetACL()

```
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	<i>Path</i>	path of file
in	<i>Options</i>	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	<i>ACL</i>	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.

9.7.2.4 hpss_GetACLHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file
in	<i>Ucred</i>	user credentials
in	<i>Options</i>	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	<i>ACL</i>	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.

9.7.2.5 hpss_SetACL()

```
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	<i>Path</i>	path of file
in	<i>Options</i>	bitmask of ACL options
in	<i>ACL</i>	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.

9.7.2.6 hpss_SetACLHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file
in	<i>Ucred</i>	user credentials
in	<i>Options</i>	bitmask of ACL options
in	<i>ACL</i>	ACL array to be set

Return values

<i>0</i>	No error.
<i>-EACCES</i>	Search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path or ACL parameter is a NULL pointer.
<i>-EINVAL</i>	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.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.

9.7.2.7 hpss_UpdateACL()

```
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	<i>Path</i>	path of file
in	<i>Options</i>	processing options
in	<i>ACL</i>	ACL array to be updated

Return values

<i>0</i>	No error.
<i>-EACCES</i>	Search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path or ACL parameter is a NULL pointer.
<i>-EINVAL</i>	There are two sets of mutually exclusive flags available through the Options paramter. An invalid combinations of flags were provided.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.

Return values

<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.

9.7.2.8 hpss_UpdateACLHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of file
in	<i>Ucred</i>	user credentials
in	<i>Options</i>	processing options
in	<i>ACL</i>	ACL array to be updated

Return values

<i>0</i>	No error.
<i>-EACCES</i>	Search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path or ACL parameter is a NULL pointer.
<i>-EINVAL</i>	There are two sets of mutually exclusive flags available through the Options paramter. An invalid combinations of flags were provided.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-EPERM</i>	The client does not have the appropriate privileges to perform the operation.

9.8 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.

9.8.1 Detailed Description

Functions used to create, remove, and manipulate directories.

9.8.2 Function Documentation

9.8.2.1 hpss_Mkdir()

```
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	<i>Path</i>	path of directory
in	<i>Mode</i>	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.

Return values

<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.

9.8.2.2 hpss_MkdirHandle()

```
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	<i>ObjHandle</i>	handle of parent directory
in	<i>Path</i>	path of directory
in	<i>Mode</i>	perm bits for new directory
in	<i>Ucred</i>	user credentials
out	<i>HandleOut</i>	returned object handle
out	<i>AttrOut</i>	returned VFS attributes

Return values

0	No error. New directory created.
<i>-EACCES</i>	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.
<i>-EEXIST</i>	The named file exists.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EMLINK</i>	The link count of the parent directory would exceed the maximum allowed number of links.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the pathname exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.

9.8.2.3 hpss_Rmdir()

```
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	<i>Path</i>	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 be overwritten by the rename, and be moved to the trashcan corresponding to the deleting user and the Path's location.

9.8.2.4 hpss_RmdirHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of directory
in	<i>Ucred</i>	user credential

Return values

<i>0</i>	Success
<i>-EACCES</i>	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.
<i>-EBUSY</i>	The named directory is currently in use and cannot be removed.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	ObjHandle is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-ENOTEMPTY</i>	The named directory contains entries other than dot and dot-dot.

Note

If trashcans are enabled, Path will be overwritten by the rename, and be moved to the trashcan corresponding to the deleting user and the Path's location.

9.8.2.5 hpss_RmdirHandleImmediate()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of directory
in	<i>Ucred</i>	user credential

Return values

<i>0</i>	Success
<i>-EACCES</i>	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.
<i>-EBUSY</i>	The named directory is currently in use and cannot be removed.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	ObjHandle is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.

Return values

<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-ENOTEMPTY</i>	The named directory contains entries other than dot and dot-dot.

Note

This directory will immediately be deleted, regardless of trashcan settings.

9.8.2.6 `hpss_RmdirImmediate()`

```
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

<i>in</i>	<i>Path</i>	path of directory

Return values

<i>0</i>	Success
<i>-EACCES</i>	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.
<i>-EBUSY</i>	The named directory is currently in use and cannot be removed.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named file does not exist, or the Path argument points to an empty string.
<i>-ENOTDIR</i>	A component of the Path prefix is not a directory.
<i>-ENOTEMPTY</i>	The named directory contains entries other than dot and dot-dot.

Note

This directory will immediately be deleted, regardless of trashcan settings.

9.9 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_GetdentsAttr](#) (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_ReadAttr](#) (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_ReadAttrHandle](#) (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_ReadAttrPlus](#) (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_ReadRawAttrHandle](#) (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.

9.9.1 Detailed Description

Functions used to access directory listings.

9.9.2 Function Documentation

9.9.2.1 hpss_Closedir()

```
int hpss_Closedir (
    int Dirdes )
```

Close an open directory stream.

Parameters

in	<i>Dirdes</i>	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

9.9.2.2 hpss_Getdents()

```
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	<i>ObjHandle</i>	directory object handle
in	<i>OffsetIn</i>	directory position
in	<i>Ucred</i>	user credentials
in	<i>BufferSize</i>	size of output buffer
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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.

Return values

<i>-EFAULT</i>	The <i>DirentPtr</i> , <i>End</i> or <i>OffsetOut</i> parameter is a NULL pointer.
<i>-EINVAL</i>	The <i>ObjHandle</i> pointer is NULL or the <i>BufferSize</i> parameter is zero.

Deprecated This function will be removed in a future version of HPSS. Use [hpss_ReaddirPlus\(\)](#).

9.9.2.3 hpss_GetdentsAttrs()

```
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	<i>ObjHandle</i>	directory object handle
in	<i>OffsetIn</i>	directory position
in	<i>Ucred</i>	user credentials
in	<i>BufferSize</i>	size of output buffer
in	<i>GetAttributes</i>	get object attributes?
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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	Success
<i>-EBADF</i>	The specified directory descriptor does not refer to an open directory.
<i>-EBUSY</i>	The directory is currently in use by another client thread.
<i>-EFAULT</i>	The <i>DirentPtr</i> , <i>End</i> or <i>OffsetOut</i> parameter is a NULL pointer.
<i>-EINVAL</i>	The <i>ObjHandle</i> pointer is NULL or the <i>BufferSize</i> parameter is zero.

Deprecated This function will be removed in a future version of HPSS. Use [hpss_ReadAttrsPlus\(\)](#).

9.9.2.4 hpss_Opendir()

```
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	<i>DirName</i>	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.

9.9.2.5 hpss_OpendirHandle()

```
int hpss_OpendirHandle (
    const ns_ObjHandle_t * DirHandle,
    const sec_cred_t * Ucred )
```

Open a directory via a handle.

Parameters

in	<i>DirHandle</i>	Directory Handle
in	<i>Ucred</i>	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.

9.9.2.6 hpss_ReadAttrs()

```
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	<i>Dirdes</i>	open directory stream handle
in	<i>OffsetIn</i>	directory position
in	<i>BufferSize</i>	size of output buffer
in	<i>GetAttributes</i>	get object attributes
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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\(\)](#).

9.9.2.7 hpss_ReadAttrsHandle()

```
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	<i>ObjHandle</i>	directory object handle
in	<i>OffsetIn</i>	directory position
in	<i>Ucred</i>	user credentials
in	<i>BufferSize</i>	size of output buffer
in	<i>GetAttributes</i>	get object attributes
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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\(\)](#).

9.9.2.8 hpss_ReadAttrsPlus()

```

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	<i>Dirdes</i>	open directory stream handle
in	<i>OffsetIn</i>	directory position
in	<i>BufferSize</i>	size of output buffer
in	<i>Flags</i>	readdir flags
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	directory entry information

Return values

>0	Number of 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.

9.9.2.9 hpss_Readdir()

```
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	<i>Dirdes</i>	open directory stream handle
out	<i>DirentPtr</i>	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 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.

9.9.2.10 hpss_ReaddirHandle()

```
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	<i>ObjHandle</i>	directory object handle
in	<i>OffsetIn</i>	directory position
in	<i>Ucred</i>	user credentials
in	<i>BufferSize</i>	size of output buffer
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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.

Return values

<i>-EFAULT</i>	One of the End, OffsetOut, or DirentPtr parameters is NULL.
<i>-EINVAL</i>	BufferSize is zero or ObjHandle is NULL.

Deprecated This function will be removed in a future version of HPSS. Use [hpss_ReaddirPlus\(\)](#).

9.9.2.11 hpss_ReaddirPlus()

```
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	<i>Dirdes</i>	Open directory descriptor
in	<i>OffsetIn</i>	directory position
in	<i>BufferSize</i>	size of output buffer
in	<i>Flags</i>	readdir flags
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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

<i>>0</i>	Number of entries read from the Core Server
<i>==0</i>	Hit end of directory
<i>-EBADF</i>	The specified directory descriptor does not refer to an open directory.
<i>-EBUSY</i>	The directory is currently in use by another client thread.
<i>-EFAULT</i>	One of the End, OffsetOut, or DirentPtr parameters is NULL.
<i>-EINVAL</i>	BufferSize is zero or ObjHandle is NULL.

9.9.2.12 hpss_ReadRawAttrsHandle()

```
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	<i>ObjHandle</i>	directory object handle
in	<i>OffsetIn</i>	directory position
in	<i>Ucred</i>	user credentials
in	<i>BufferSize</i>	size of output buffer
in	<i>GetAttributes</i>	get object attributes
out	<i>End</i>	hit end of directory
out	<i>OffsetOut</i>	resulting directory position
out	<i>DirentPtr</i>	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\(\)](#).

9.9.2.13 hpss_Rewinddir()

```
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.

9.10 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.

9.10.1 Detailed Description

Functions used to determine or modify the working directory.

9.10.2 Function Documentation

9.10.2.1 hpss_Chdir()

```
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	<i>Path</i>	Path to the directory

Return values

0	No error. New working directory set properly.
-EACCES	Search permission is denied on a component of the <i>Path</i> prefix.
-EFAULT	The <i>Path</i> parameter is a NULL pointer.
-ENAMETOOLONG	The length of the <i>Path</i> 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 <i>Path</i> argument points to an empty string.
-ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

9.10.2.2 hpss_Chroot()

```
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	<i>Path</i>	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.

Return values

<i>-ENOMEM</i>	Memory allocation failure

Note

As currently implemented, symbolic links allow a client to access files outside the new root directory.

9.10.2.3 **hpss_Getcwd()**

```
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	<i>Buf</i>	current working directory
in	<i>Size</i>	size of the buffer

Return values

0	No error.
<i>-EACCES</i>	Read or Search permission is denied on a component of the path name.
<i>-EFAULT</i>	The Buf parameter is a NULL pointer.
<i>-EINVAL</i>	The Size argument is zero.
<i>-ERANGE</i>	The Size argument is greater than zero, but smaller than the length of the path name plus 1.

9.10.2.4 **hpss_GetThreadUcred()**

```
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	<i>RetUcred</i>	credentials on client

Return values

0	No error. The RetUcred contains the current credentials for the user.
-EFAULT	RetUcred is a NULL pointer.

9.11 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.

9.11.1 Detailed Description

Functions used to control Client API state.

9.11.2 Function Documentation

9.11.2.1 API_GetDataThreadStackSize()

```
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](#)

9.11.2.2 API_SocketSetOptions()

```
void API_SocketSetOptions (
    int Socket,
    hpss_sockaddr_t * SaddrPtr,
    unsigned int Options )
```

Sets the API socket options.

Parameters

in	<i>Socket</i>	Socket descriptor
in	<i>SaddrPtr</i>	Socket address
in	<i>Options</i>	Socket options

The API_SocketSetOptions routine is called to set the send and receive buffer sizes, TCP no delay and keepalive flags.

Return values

<i>None.</i>	

9.11.2.3 hpss_ClientAPIInit()

```
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
-ENOCNNECT	Could not initialize HPSS connection services.
-ENOMEM	Memory allocation failed.
-EPERM	Could not initialize security.

9.11.2.4 hpss_ClientAPIReset()

```
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](#)

9.11.2.5 hpss_GetConfiguration()

```
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.

9.11.2.6 hpss_LoadDefaultThreadState()

```
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	<i>UserID</i>	New user credentials
in	<i>Umask</i>	New umask
in	<i>ClientFullName</i>	Client fully qualified name

Return values

<i>-EAGAIN</i>	Default security registry error.
<i>-EINVAL</i>	NumGroups is invalid
<i>-ENOCONECT</i>	Security registry is unavailable.
<i>-ENOENT</i>	No thread with given ThreadID or security registry entry was not found.
<i>-ENOMEM</i>	Unable to allocate memory
<i>0</i>	No error. Thread specific state is valid.

Note

The effects of this call aren't reversed by resetting the API.

9.11.2.7 hpss_MiscAdmin()

```
int hpss_MiscAdmin (
    uint32_t Function,
    char * TextP,
    ns_AdminConfArray_t * ValueConfArrayP )
```

Accesses the Core Server's miscellaneous functions.

Parameters

in	<i>Function</i>	Func to perform.
in, out	<i>TextP</i>	For auxil text.
in, out	<i>ValueConfArrayP</i>	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 logging 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 ValueConfArray 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

9.11.2.8 hpss_SetAPILogLevel()

```
int hpss_SetAPILogLevel (
    int LogLevel )
```

Modifies the Client API log level.

Parameters

in	<i>LogLevel</i>	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

9.11.2.9 hpss_SetAPILogPath()

```
int hpss_SetAPILogPath (
    const char * LogPath )
```

Modifies the Client API log path.

Parameters

in	<i>LogPath</i>	Client API Log Path

Return values

0	Success
<i>EACCES</i>	Requested access is not allowed
<i>EFAULT</i>	LogPath is outside addressable address space
<i>EISDIR</i>	LogPath refers to a directory
<i>ELOOP</i>	Too many symbolic links were encountered
<i>EMFILE</i>	The process already has the maximum number of files open.
<i>ENAMETOOLONG</i>	LogPath is too long.
<i>ENOSPC</i>	No space for the new logfile
<i>ENOTDIR</i>	A component used as a directory in LogPath was not a directory
<i>EOVERFLOW</i>	LogPath refers to a regular file that is too large to be opened
<i>EROFS</i>	LogPath refers to a file on a read-only file system
<i>ENOMEM</i>	Ran out of memory

9.11.2.10 hpss_SetApplicationName()

```
void hpss_SetApplicationName (
    const char * App )
```

Modifies the Client API application name.

Parameters

in	<i>App</i>	Application name string

9.11.2.11 hpss_SetConfiguration()

```
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	<i>ConfigIn</i>	Configuration parameters

Return values

0	Success
-EFAULT	The ConfigIn parameter is NULL
-EINVAL	Invalid configuration attribute value setting

9.11.2.12 hpss_ThreadCleanUp()

```
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

9.12 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.

9.12.1 Detailed Description

Functions used to authenticate the Client API with HPSS.

9.12.2 Function Documentation

9.12.2.1 hpss_PurgeLoginCred()

```
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.

9.12.2.2 hpss_SetAppLoginCred()

```
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	<i>AppName</i>	Application name
in	<i>PrincipalName</i>	Principal name
in	<i>Mechanism</i>	Security Mechanism
in	<i>CredType</i>	Type of creds needed
in	<i>AuthType</i>	Authenticator type
in	<i>Authenticator</i>	Authenticator

Return values

0	Success
<i>Otherwise</i>	An other HPSS errno indicating nature 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 credential. This function is not thread safe, and they should only be called within a single thread, or serialized using a mutex.

9.12.2.3 hpss_SetAuditInfo()

```

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	<i>EndUserHostAddr</i>	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

9.12.2.4 hpss_SetLoginCred()

```
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	<i>PrincipalName</i>	Principal name
in	<i>Mechanism</i>	Security Mechanism
in	<i>CredType</i>	Type of creds needed
in	<i>AuthType</i>	Authenticator type
in	<i>Authenticator</i>	Authenticator

Return values

0	Success
Otherwise	An other HPSS errno indicating nature 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 credential. This function is not thread safe, and they should only be called within a single thread, or serialized using a mutex.

9.13 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.

9.13.1 Detailed Description

Functions used to retrieve or reset HPSS statistics.

9.13.2 Function Documentation

9.13.2.1 hpss_GetSubSysLimits()

```
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	<i>SubsystemID</i>	Identifier for the subsystem
in	<i>Ucred</i>	User Credentials
out	<i>LimitsOut</i>	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.

9.13.2.2 hpss_GetSubSysStats()

```
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	<i>SubsystemID</i>	identifier for the subsystem
out	<i>StatsOut</i>	subsystem statistics information

Return values

0	No error.
-EFAULT	The StatsOut parameter is a NULL pointer.

9.13.2.3 hpss_ResetSubSysStats()

```
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	<i>SubsystemID</i>	identifier for the subsystem
out	<i>StatsOut</i>	subsystem statistics information

Return values

0	No error.
-EFAULT	The StatsOut parameter is a NULL pointer.

9.13.2.4 hpss_Statfs()

```
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 LS 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	<i>CosId</i>	Class of service id.
out	<i>StatfsBuffer</i>	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\(\)](#).

9.13.2.5 hpss_Statfs64()

```
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 LS 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	<i>CosId</i>	Class of service id.
out	<i>StatfsBuffer</i>	file system status.

Return values

0	Success
-EFAULT	The StatfsBuffer parameter is a NULL pointer.
-EINVAL	The specified Class of Service does not exist.

9.13.2.6 hpss_Statvfs()

```
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 LS 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	<i>CosId</i>	Class of service id.
out	<i>StatvfsBuffer</i>	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.

9.13.2.7 hpss_Statvfs64()

```
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 LS 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	<i>CosId</i>	Class of service id.
out	<i>StatvfsBuffer</i>	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.

9.14 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 *FilesetRootHandle, char *Path)
Retrieves the pathname and root fileset that correspond to an Object and Subsystem.

9.14.1 Detailed Description

Functions used to manipulate or construct HPSS objects.

9.14.2 Function Documentation

9.14.2.1 hpss_GetObjFromHandle()

```
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	<i>ObjHandle</i>	parent object handle
in, out	<i>ObjHash</i>	object id / hash pair

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>-EFAULT</i>	The <i>ObjHandle</i> parameter is a NULL pointer.

9.14.2.2 **hpss_GetObjType()**

```
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	<i>ObjHandle</i>	parent object handle

Return values

<i>-EFAULT</i>	The <i>ObjHandle</i> parameter is a NULL pointer.
<i>-EINVAL</i>	The <i>ObjHandle</i> points to an invalid object.
<i>other</i>	Object type

9.14.2.3 **hpss_GetPathHandle()**

```
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	<i>ObjHandle</i>	object handle
out	<i>FilesetRootHandle</i>	Fileset root handle
out	<i>Path</i>	Path to object

Return values

<i>-EFAULT</i>	The Path or FilesetRootHandle parameter is NULL.
<i>-EINVAL</i>	The ObjHandle is NULL.

9.14.2.4 **hpss_GetPathHandleObjHash()**

```
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	<i>ObjIdHash</i>	object hash
in	<i>SubsysId</i>	subsystem id
out	<i>FilesetRootHandle</i>	Fileset root handle
out	<i>ObjHandle</i>	object handle
out	<i>Path</i>	Path to object

Return values

0	Success
<i>-EFAULT</i>	Path pointer is NULL
<i>-ENAMETOOLONG</i>	The path name could not be retrieved because it is too long.
<i>-ENOENT</i>	The path could not be retrieved because no path

9.14.2.5 **hpss_GetPathObjHash()**

```
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	<i>ObjIdHash</i>	object id / hash pair
in	<i>SubsysId</i>	subsystem id
out	<i>FilesetRootHandle</i>	Fileset root handle
out	<i>Path</i>	Path to object

Return values

<i>-EFAULT</i>	Path pointer is NULL
<i>-ENAMETOOLONG</i>	The path name could not be retrieved because it is too long.
<i>-ENOENT</i>	The path could not be retrieved because no path corresponds to the given object id.

9.15 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.

9.15.1 Detailed Description

Functions used to read and write data using a stream buffer.

9.15.2 Function Documentation

9.15.2.1 `hpss_Fclose()`

```
int hpss_Fclose (
    HPSS_FILE * hpss_Stream )
```

Closes a stream and cleans up.

Parameters

<code>in, out</code>	<code>hpss_Stream</code>	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

<code>0</code>	No error. The stream is closed.
<code>-EACCES</code>	Permission denied.
<code>-EBADF</code>	The file descriptor underlying the Stream is not valid.
<code>-EBUSY</code>	Some other thread is manipulating this file.
<code>-EINVAL</code>	Invalid (NULL) <code>hpss_Stream</code> parameter.
<code>-EIO</code>	An internal error occurred.

Note

Resources Used: Memory for I/O buffers and `HPSS_FILE` structure.

See also

[hpss_Fflush](#)

9.15.2.2 `hpss_Fcntl()`

```
int hpss_Fcntl (
    int hpss_fd,
    int Cmd,
    long Arg )
```

Set or get file control arguments.

Parameters

in	<i>hpss↵ _fd</i>	HPSS file descriptor
in	<i>Cmd</i>	Requested command
in	<i>Arg</i>	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

<i>-EBUSY</i>	Filetable entry busy
<i>-EINVAL</i>	Invalid argument value
<i>-EIO</i>	Internal I/O error
<i>-EFAULT</i>	Invalid object handle
<i>-ENOTSUP</i>	Command not supported
<i>-EBADF</i>	Bad file descriptor
<i>-ENOMEM</i>	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.

9.15.2.3 hpss_Fflush()

```
int hpss_Fflush (
    HPSS_FILE * hpss_Stream )
```

Writes buffered data for a stream.

Parameters

<i>in, out</i>	<i>hpss_Stream</i>	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

<i>0</i>	No error. The buffer was flushed.
<i>-EACCES</i>	Permission denied.
<i>-EBADF</i>	The file descriptor underlying the Stream is not valid.
<i>-EINVAL</i>	Invalid (NULL) <i>hpss_Stream</i> parameter.
<i>-ESTALE</i>	Stale connections
<i>-EIO</i>	An internal error occurred.

See also

[hpss_Fopen](#) [hpss_Fclose](#)

9.15.2.4 hpss_Fgetc()

```
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

<i>in, out</i>	<i>stream</i>	stream from which to receive data

Return values

<i>Positive</i>	Character read from the data stream
<i>EOF</i>	End of file reached.
<i>-EBADF</i>	Invalid file descriptor or file not open.
<i>-EINVAL</i>	Invalid input parameter (Ptr Null or Size zero)
<i>-EFAULT</i>	NULL stream pointer.

Return values

-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.

9.15.2.5 hpss_Fgets()

```
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	<i>s</i>	Buffer in which to receive data
in	<i>n</i>	number of bytes to read
in	<i>stream</i>	stream from which to receive data

Return values

<i>Not</i>	Null String Data In Buffer
<i>Null</i>	String End of File or Error Reached
<i>EBUSY</i>	Errno may be set to EBUSY if the HPSS file table entry is busy.
<i>ESTALE</i>	Errno may be set to ESTALE if the connection to HPSS has become stale.
<i>EPERM</i>	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.
<i>EIO</i>	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.

9.15.2.6 hpss_Fopen()

```
HPSS_FILE* hpss_Fopen (
    const char * Path,
    const char * Mode )
```

Opens an HPSS file and associates a stream with it.

Parameters

in	<i>Path</i>	Path/name of file
in	<i>Mode</i>	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

<i>EFAULT</i>	Path parameter is NULL
<i>EINVAL</i>	Invalid mode parameter
<i>EIO</i>	Internal error
<i>ENOENT</i>	Path parameter points to an empty string
<i>ENOMEM</i>	Memory allocation failure
<i>EMFILE</i>	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.

9.15.2.7 hpss_Fread()

```
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	<i>Ptr</i>	Pointer to input array.
in	<i>Size</i>	Size in bytes of data items
in	<i>Num</i>	Number of data items to read
in, out	<i>hpss_Stream</i>	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 occurred.
ENOENT	File doesn't exist for reading.
ENOMEM	Memory allocation failure.

9.15.2.8 hpss_Fseek()

```
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

<code>-EINVAL</code>	Invalid input parameter
<code>-EFAULT</code>	Invalid or NULL pointer
<code>-EBADF</code>	HPSS File not opened
<code>-EBUSY</code>	HPSS File table entry busy
<code>-ESTALE</code>	Connection has gone stale

Note

The behavior of this routine is modeled after the system `fseek()`

9.15.2.9 hpss_Fsync()

```
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	<code>hpss↔ _fd</code>	valid HPSS file descriptor

Return values

<code>0</code>	<code>hpss_fd</code> is a valid file descriptor.
<code>-EBADF</code>	<code>hpss_fd</code> is not a valid file descriptor.

9.15.2.10 hpss_Ftell()

```
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	<i>hpss_Stream</i>	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

9.15.2.11 **hpss_Fwrite()**

```
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	<i>Ptr</i>	Pointer to input array.
in	<i>Size</i>	Size in bytes of data items
in	<i>Num</i>	Number of data items to write
in, out	<i>hpss_Stream</i>	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 occurred.
-ENOENT	File doesn't exist for writing.
-ENOMEM	Memory allocation failure.
-ESTALE	Connection not valid for this file table entry.

9.16 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 which 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_UserAttrSetAttr](#) (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.

9.16.1 Detailed Description

Functions used to update and access user-defined metadata.

9.16.2 Function Documentation

9.16.2.1 hpss_UserAttrCloseCursor()

```
int hpss_UserAttrCloseCursor (
    int SubsystemId,
    const hpss_cursor_id_t * CursorId )
```

Close a cursor.

Parameters

in	<i>SubsystemId</i>	Subsystem Id
in	<i>CursorId</i>	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.

Return values

<i>-EFAULT</i>	The CursorId parameter is NULL.
<i>-ESRCH</i>	The cursor requested to be closed could not be found and may have already been closed due to a timeout.

9.16.2.2 hpss_UserAttrDeleteAttrHandle()

```
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	<i>Obj</i>	Parent Object
in	<i>Path</i>	Namespace Path
in	<i>Ucred</i>	user credentials
in	<i>Attrs</i>	Attributes
in	<i>Schema</i>	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

<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	One of the attribute pairs in Attr is invalid.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	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.

9.16.2.3 hpss_UserAttrDeleteAttrs()

```
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	<i>Path</i>	Namespace Path
in	<i>Attrs</i>	Attributes
in	<i>Schema</i>	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.

9.16.2.4 hpss_UserAttrGetAttrHandle()

```
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	<i>Obj</i>	Parent Object
in	<i>Path</i>	Namespace Path
in	<i>Ucred</i>	user credentials
out	<i>Attrs</i>	Attributes and Values
in	<i>XMLFlag</i>	XML Flag
in	<i>XMLSize</i>	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

<i>-EACCES</i>	The user does not have permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer and Obj is NULL.
<i>-EINVAL</i>	One of the attribute pairs in Attr is invalid.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named object does not exist, or the Path argument points to an empty string.
<i>-ENOATTR</i>	No user attributes exist for the specified object.
<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-ERANGE</i>	The length of the data was too large to be returned completely.
<i>-EDOM</i>	The XMLSize was too large.

9.16.2.5 hpss_UserAttrGetAttrs()

```
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	<i>Path</i>	Namespace Path
out	<i>Attrs</i>	Attributes and Values
in	<i>XMLFlag</i>	XML Flag
in	<i>XMLSize</i>	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

<i>-EACCES</i>	The user does not have read permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	One of the attribute pairs in Attr is invalid, or an invalid XMLFlag was used.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named object does not exist, or the Path argument points to an empty string.
<i>-ENOATTR</i>	No user attributes exist for the specified object.

Return values

-ERANGE	The length of the data was too large to be returned completely.
-EDOM	The XMLSize was too large.

9.16.2.6 hpss_UserAttrGetObj()

```
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	<i>SubsystemId</i>	Subsystem Id
in	<i>Attrs</i>	Attribute names
in	<i>MaxObjs</i>	Max objs to return
in, out	<i>ObjectList</i>	List of Id / Hash Pairs
in, out	<i>CursorId</i>	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.

9.16.2.7 hpss_UserAttrGetObjQuery()

```
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	<i>SubsystemId</i>	Subsystem Id
in	<i>MaxObjs</i>	Max objs to return
out	<i>ObjectList</i>	Objects Found
in	<i>Query</i>	Query to Execute
in, out	<i>CursorId</i>	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.

9.16.2.8 hpss_UserAttrGetXML()

```

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	<i>SubsystemId</i>	Subsystem Id
in	<i>MaxXMLs</i>	Max Entries to Return
out	<i>XML</i>	XML Returned

Parameters

in	<i>XQuery</i>	XQuery
out	<i>CursorId</i>	Optional CursorId
in	<i>XMLSize</i>	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.

9.16.2.9 hpss_UserAttrGetXMLObj()

```
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 which 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	<i>SubsystemId</i>	Subsystem Id
in	<i>MaxXMLObjs</i>	Max XMLObjs to return
out	<i>ObjectList</i>	Object List
out	<i>XML</i>	XML List
in	<i>XPathWhere</i>	'Where' XPath
in	<i>XPathFind</i>	'Find' XPath
in, out	<i>CursorId</i>	CursorId
in	<i>XMLSize</i>	XML Size

Note

Cursors used by this function will expire after a certain amount of time if they are idle.

Return values

<i>-EACCES</i>	The user is not a Trusted User with Control Permission in the Core Server.
<i>-EFAULT</i>	The XQuery, ObjectList, or XML parameters are NULL.
<i>-EINVAL</i>	MaxXMLObjs is an invalid number; XPathFind or XPathWhere is invalid.
<i>-ENOENT</i>	There were no entries which matched the query.
<i>-ERANGE</i>	One or more of the results contained more data than could be returned.

9.16.2.10 hpss_UserAttrListAttrs()

```
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	<i>Path</i>	Namespace Path
out	<i>Attrs</i>	Attributes
in	<i>Flags</i>	Flags
in	<i>XMLSize</i>	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
<i>-EACCES</i>	The user does not have read permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	The XML from Path could not be converted into Attribute format.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	The named object does not exist, or the Path argument points to an empty string.
<i>-ENOATTR</i>	No user attributes exist for the specified object.
<i>-ERANGE</i>	The length of the data requested was too large to be returned.
<i>-EDOM</i>	The XMLSize was too large.

Warning

The attributes returned by this function must be freed by the caller.

9.16.2.11 hpss_UserAttrReadObj()

```
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	<i>SubsystemId</i>	Subsystem Id
out	<i>ObjectList</i>	Objects
in	<i>CursorId</i>	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.

9.16.2.12 hpss_UserAttrReadXML()

```
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	<i>SubsystemId</i>	Subsystem Id
out	<i>XML</i>	XML Returned
in	<i>CursorId</i>	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.

9.16.2.13 hpss_UserAttrReadXMLObj()

```
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	<i>SubsystemId</i>	Subsystem Id
out	<i>ObjectList</i>	Objects
out	<i>XML</i>	XML Returned
in	<i>CursorId</i>	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.

9.16.2.14 hpss_UserAttrSetAttrHandle()

```
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	<i>Obj</i>	Parent Object
in	<i>Path</i>	Namespace Path
in	<i>Ucred</i>	user credentials
in	<i>Attrs</i>	Attributes
in	<i>Schema</i>	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.

9.16.2.15 hpss_UserAttrSetAttrs()

```

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	<i>Path</i>	Namespace Path
in	<i>Attrs</i>	Attributes
in	<i>Schema</i>	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.

9.16.2.16 hpss_UserAttrXQueryGet()

```
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	<i>Path</i>	Namespace Path
in	<i>XQuery</i>	XQuery
in, out	<i>Buffer</i>	Buffer
in	<i>XMLSize</i>	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.
-ENOATTR	No user attributes exist for the specified object.
-ERANGE	The requested data was too large to be returned completely.

9.16.2.17 hpss_UserAttrXQueryGetHandle()

```
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	<i>Obj</i>	Parent Object
in	<i>Path</i>	Namespace Path
in	<i>Ucred</i>	user credentials
in	<i>XQuery</i>	XQuery
in, out	<i>Buffer</i>	Buffer
in	<i>XMLSize</i>	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.
-ENOATTR	No user attributes exist for the specified object.
-ERANGE	The requested data was too large to be returned completely.

9.16.2.18 hpss_UserAttrXQueryUpdate()

```
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	<i>Path</i>	Namespace Path
in	<i>XQuery</i>	XQuery string
in	<i>Schema</i>	Schema

Return values

<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	Invalid XQuery.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	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.

9.16.2.19 `hpss_UserAttrXQueryUpdateHandle()`

```
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	<i>Obj</i>	Namespace Object
in	<i>Path</i>	Namespace Path
in	<i>Ucred</i>	user credentials
in	<i>XQuery</i>	XQuery
in	<i>Schema</i>	Schema

Return values

<i>-EACCES</i>	The user does not have write permission on the namespace object, or search permission is denied on a component of the path prefix.
<i>-EFAULT</i>	The Path parameter is a NULL pointer.
<i>-EINVAL</i>	Invalid XQuery.
<i>-ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>-ENOENT</i>	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 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.

9.17.1 Detailed Description

Functions used to convert HPSS constructs to POSIX equivalents.

9.17.2 Function Documentation

9.17.2.1 API_CalcBlock32()

```
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	<i>Attrs</i>	HPSS object attributes
out	<i>Blksize</i>	Block Size
out	<i>Numblks</i>	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.

9.17.2.2 API_CalcBlock64()

```
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	<i>Attrs</i>	HPSS object attributes
out	<i>Blksize</i>	Block Size
out	<i>Numblks</i>	Number of Blocks

See also

[API_CalcBlock32](#), [API_ConvertModeToPosixMode](#), [API_ConvertPosixModeToMode](#), [API_ConvertPosixTimeToTime](#),
[API_ConvertTimeToPosixTime](#)

Note

The block size and number of blocks will use the HPSS 64-bit type.

9.17.2.3 API_ConvertModeToPosixMode()

```
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	<i>Attrs</i>	HPSS object attributes
out	<i>PosixMode</i>	Pointer to posix mode_t

See also

[API_CalcBlock32](#), [API_CalcBlock64](#), [API_ConvertPosixModeToMode](#), [API_ConvertPosixTimeToTime](#),
[API_ConvertTimeToPosixTime](#)

9.17.2.4 API_ConvertPosixModeToMode()

```
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	<i>PosixMode</i>	POSIX mode_t to be translated
out	<i>Attrs</i>	Pointer to HPSS attributes

See also

[API_CalcBlock32](#), [API_CalcBlock64](#), [API_ConvertModeToPosixMode](#), [API_ConvertPosixTimeToTime](#),
[API_ConvertTimeToPosixTime](#)

9.17.2.5 API_ConvertPosixTimeToTime()

```
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	<i>Attrs</i>	HPSS object attributes
in	<i>Atime</i>	Last Access time
in	<i>Mtime</i>	Last Modify time
in	<i>Ctime</i>	Last Change time

See also

[API_CalcBlock32](#), [API_CalcBlock64](#), [API_ConvertModeToPosixMode](#) [API_ConvertPosixModeToMode](#),
[API_ConvertTimeToPosixTime](#)

9.17.2.6 API_ConvertTimeToPosixTime()

```
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	<i>Attrs</i>	HPSS object attributes
out	<i>Atime</i>	Last Access time
out	<i>Mtime</i>	Last Modify time
out	<i>Ctime</i>	Last Change time

See also

[API_CalcBlock32](#), [API_CalcBlock64](#), [API_ConvertModeToPosixMode](#) [API_ConvertPosixModeToMode](#),
[API_ConvertPosixTimeToTime](#)

9.18 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 *Data↔Buffer, 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.

9.18.1 Detailed Description

Functions which implement HPSS parallel I/O capabilities.

9.18.2 Function Documentation

9.18.2.1 [hpss_PIOEnd\(\)](#)

```
int hpss_PIOEnd (
    hpss\_pio\_grp\_t StripeGroup )
```

End and clean up a parallel IO group context.

Parameters

in, out	<i>StripeGroup</i>	IO stripe group

This function is used by a transfer coordinator or participant to end a parallel IO group context.

Return values

<i>-ENOMEM</i>	Can't allocate memory for stack entry
<i>-EINVAL</i>	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.

9.18.2.2 hpss_PIOExecute()

```
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	<i>Fd</i>	Open file descriptor
in	<i>FileOffset</i>	IO file offset
in	<i>Size</i>	IO size
in	<i>StripeGroup</i>	IO stripe group
out	<i>GapInfo</i>	Sparse file information
out	<i>BytesMoved</i>	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.

9.18.2.3 hpss_PIOExecuteItem()

```
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	<i>Fd</i>	Open file descriptor
in	<i>FileOffset</i>	IO file offset
in	<i>Size</i>	IO size
in	<i>StripeGroup</i>	IO stripe group
out	<i>GapInfo</i>	Sparse file information
out	<i>BytesMoved</i>	Bytes moved
in	<i>Item</i>	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.

9.18.2.4 hpss_PIOExportGrp()

```

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	<i>StripeGroup</i>	IO stripe group
out	<i>Buffer</i>	Raw data buffer
out	<i>BufLength</i>	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

<i>-ENOMEM</i>	Can't allocate memory for stack entry
<i>-EFAULT</i>	Buffer or Buffer length were NULL.
<i>-EINVAL</i>	Group context was NULL or could not be verified.
<i>0</i>	otherwise

Note

This function runs in the coordinator context/role.

9.18.2.5 hpss_PIOFinalize()

```
int hpss_PIOFinalize (
    hpss_pio_grp_t * StripeGroup )
```

Safely end and clean up a parallel IO group context.

Parameters

in	<i>StripeGroup</i>	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

<i>-EFAULT</i>	NULL stripe group provided
<i>0</i>	Success
<i>Other</i>	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.

9.18.2.6 hpss_PIOGetPartError()

```
int hpss_PIOGetPartError (
    hpss_pio_grp_t StripeGroup,
    int * Error )
```

Retrieves a participant error code.

Parameters

<i>in</i>	<i>StripeGroup</i>	IO stripe group
<i>in, out</i>	<i>Error</i>	Participant Error Code

This function is called to retrieve the participant error code from the stripe group context.

Return values

<i>0</i>	Success
<i>-EFAULT</i>	No group supplied
<i>-EINVAL</i>	No error supplied

9.18.2.7 hpss_PIOGetRequestID()

```
int hpss_PIOGetRequestID (
    hpss_pio_grp_t StripeGroup,
    hpss_reqid_t * RequestID )
```

Retrieves the request ID for a group.

Parameters

<i>in</i>	<i>StripeGroup</i>	IO stripe group
<i>in, out</i>	<i>RequestID</i>	Coordinator Request ID

This function is called to retrieve the coordinator's request id from the stripe group context.

Return values

<i>0</i>	Success
<i>-EFAULT</i>	No group supplied
<i>-EINVAL</i>	No request ID supplied

Note

For other I/O, use `hpss_GetNextIORequestID` to obtain the request id.

9.18.2.8 hpss_PIOImportGrp()

```
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	<i>Buffer</i>	Raw data buffer
in	<i>BufLength</i>	Raw buffer length
out	<i>StripeGroup</i>	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

<i>-ENOMEM</i>	Can't allocate memory for stack entry
<i>-EINVAL</i>	StripeGroup or Buffer is NULL
<i>0</i>	otherwise

Note

This function runs in the participant context/role.

9.18.2.9 hpss_PIORegister()

```
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	<i>StripeElement</i>	Element of the stripe
in	<i>DataNetSockAddr</i>	NIC address for data movement
in	<i>DataBuffer</i>	Data buffer for IO
in	<i>DataBufLen</i>	Data buffer length
in, out	<i>StripeGroup</i>	IO stripe group
in	<i>IOCallback</i>	IO call back function
in	<i>IOCallbackArg</i>	IO call back argument

This is used by a transfer stripe participant to register an IO call back function.

Return values

<i>0</i>	Success
<i>-EINVAL</i>	Invalid argument or checksum.

Return values

<code>-ENOMEM</code>	Not enough memory to allocate stripe group.

Note

This function runs in the participant context/role.

9.18.2.10 hpss_PIOStart()

```
int hpss_PIOStart (
    hpss_pio_params_t * InputParams,
    hpss_pio_grp_t * StripeGroup )
```

Start a new parallel I/O group context.

Parameters

<code>in, out</code>	<i>InputParams</i>	Input parameters
<code>out</code>	<i>StripeGroup</i>	New IO stripe group

This function is used by a transfer coordinator to start a new parallel IO group context.

Return values

<code>0</code>	Success
<code>-EFAULT</code>	StripeGroup provided was NULL.
<code>-EINVAL</code>	InputParams provided was NULL, invalid operation specified, or zero block size provided.
<code>-ENOMEM</code>	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.

9.19 Simplified Parallel I/O

Functions which wrap PIO for ease of use with local file descriptors.

Functions

- int `hpss_PIOInitCtx` (hpss_pio_t *p)
Start a simplified PIO context.

- `int hpss_PIOReadFile` (`int hpss_fd`, `uint64_t data_length`, `int local_fd`, `uint64_t *bytes_copied`, `hpss_read_queue_item_t *item`)
Read a file from HPSS to a local file using PIO.
- `int hpss_PIOReadFileMulti` (`hpss_pio_t *Ctx`, `int hpss_fd`, `uint64_t data_length`, `int local_fd`, `uint64_t *bytes_copied`, `hpss_read_queue_item_t *item`)
Issue a simplified PIO read.
- `int hpss_PIOResetCtx` (`hpss_pio_t *p`)
Stop a simplified PIO context.
- `int hpss_PIOSetBlockSize` (`hpss_pio_t *Ctx`, `uint32_t BlockSize`)
Set the client transfer block size.
- `int hpss_PIOSetDefaultCtx` (`hpss_pio_t *Ctx`)
Set default values in the PIO context.
- `int hpss_PIOSetHandleGaps` (`hpss_pio_t *Ctx`, `int HandleGaps`)
Set the client gap handling behavior.
- `int hpss_PIOSetOffset` (`hpss_pio_t *Ctx`, `uint64_t Offset`)
Set the offset in the PIO context.
- `int hpss_PIOSetStripeWidth` (`hpss_pio_t *Ctx`, `uint32_t Width`)
Set the client stripe width.
- `int hpss_PIOSetUseAPIHost` (`hpss_pio_t *Ctx`, `int UseAPIHost`)
Set the client data hostname behavior.
- `int hpss_PIOWriteFile` (`int hpss_fd`, `uint64_t data_length`, `int local_fd`, `uint64_t *bytes_copied`)
Write a file to HPSS from a local file using PIO.
- `int hpss_PIOWriteFileMulti` (`hpss_pio_t *Ctx`, `int hpss_fd`, `uint64_t data_length`, `int local_fd`, `uint64_t *bytes_copied`)
Write a file to HPSS from a local file using PIO.

9.19.1 Detailed Description

Functions which wrap PIO for ease of use with local file descriptors.

9.19.2 Function Documentation

9.19.2.1 `hpss_PIOInitCtx()`

```
int hpss_PIOInitCtx (
    hpss_pio_t * p )
```

Start a simplified PIO context.

Parameters

<code>in, out</code>	<code>Ctx</code>	Context to modify

This creates a PIO context which can be used with `hpss_PIOReadFileMulti` and `hpss_PIOWriteFileMulti`. The PIO options can be set using `hpss_PIOSetStripeWidth`, `hpss_PIOSetOffset`, and `hpss_PIOSetDefaultCtx`.

Return values

0	Success
-EINVAL	Invalid input parameters

Note

The PIO context must be reset (using `hpss_PIOResetCtx`) between calls.

9.19.2.2 `hpss_PIOReadFile()`

```
int hpss_PIOReadFile (
    int hpss_fd,
    uint64_t data_length,
    int local_fd,
    uint64_t * bytes_copied,
    hpss_read_queue_item_t * item )
```

Read a file from HPSS to a local file using PIO.

Parameters

in	<i>hpss_fd</i>	Open HPSS File Descriptor
in	<i>data_length</i>	Amount of data to read from HPSS
in	<i>local_fd</i>	Open local file descriptor to write to
out	<i>bytes_copied</i>	Number of bytes copied to local_fd
in	<i>item</i>	Read Queue Item (optional)

This function will use PIO with default settings (from `hpss_PIOSetDefaultCtx`) to copy data from the `hpss_fd` to `local_fd`, up to `data_length` bytes. If `item` is provided, it will be used to take advantage of read queuing.

Return values

0	Success
-EIO	I/O Error
-EINVAL	Invalid input parameters

9.19.2.3 `hpss_PIOReadFileMulti()`

```
int hpss_PIOReadFileMulti (
    hpss_pio_t * Ctx,
    int hpss_fd,
    uint64_t data_length,
    int local_fd,
```



```
uint64_t * bytes_copied,
hpss_read_queue_item_t * item )
```

Issue a simplified PIO read.

Parameters

in, out	<i>Ctx</i>	Context to use
in	<i>hpss_fd</i>	Open HPSS File Descriptor
in	<i>data_length</i>	Amount of data to read from HPSS
in	<i>local_fd</i>	Open local file descriptor to write to
out	<i>bytes_copied</i>	Number of bytes copied to local_fd
in	<i>item</i>	Read Queue Item (optional)

This function will use PIO with default settings (from `hpss_PIOSetDefaultCtx`) to copy data from the `hpss_fd` to `local_fd`, up to `data_length` bytes. If `item` is provided, it will be used to take advantage of read queuing.

Return values

0	Success
-EIO	I/O Error
-EINVAL	Invalid input parameters

9.19.2.4 hpss_PIOResetCtx()

```
int hpss_PIOResetCtx (
    hpss_pio_t * p )
```

Stop a simplified PIO context.

Parameters

in, out	<i>Ctx</i>	Context to modify

This resets a PIO context. This allows it to be used again with `hpss_PIOInitCtx` or a call to `hpss_PIOReadFileMulti` and `hpss_PIOWriteFileMulti`.

Return values

0	Success
-EINVAL	Invalid input parameters

9.19.2.5 hpss_PIOSetBlockSize()

```
int hpss_PIOSetBlockSize (
```

```

    hpss_pio_t * Ctx,
    uint32_t BlockSize )

```

Set the client transfer block size.

Parameters

<i>in, out</i>	<i>Ctx</i>	Context to modify
<i>in</i>	<i>Width</i>	Client stripe width

Update the block size for the context to the provided value.

Return values

<i>0</i>	Success
<i>-EINVAL</i>	Invalid input parameters

9.19.2.6 hpss_PIOSetDefaultCtx()

```

int hpss_PIOSetDefaultCtx (
    hpss_pio_t * Ctx )

```

Set default values in the PIO context.

Parameters

<i>in, out</i>	<i>Ctx</i>	Context to fill with defaults

This function applies any default values to the input context. Only values which have environment defaults configured will be changed.

This function responds to the following environment variables:

- HPSS_API_PIO_STRIPE_WIDTH (int), stripe width (must be <= HPSS_MAX_STRIPE_WIDTH)
- HPSS_API_PIO_BLOCK_SIZE (int), transfer block size
- HPSS_API_PIO_HANDLE_GAPS (1/0), handle gaps
- HPSS_API_PIO_USE_API_HOST (1/0), use the HPSS_API_HOSTNAME for the data hostname for Mover connections

Return values

<i>0</i>	Success

9.19.2.7 hpss_PIOSetHandleGaps()

```
int hpss_PIOSetHandleGaps (
    hpss_pio_t * Ctx,
    int HandleGaps )
```

Set the client gap handling behavior.

Parameters

in, out	<i>Ctx</i>	Context to modify
in	<i>HandleGaps</i>	Handle Gaps (true/false)

Update the gap handling behavior for the context to the provided value.

Return values

0	Success
-EINVAL	Invalid input parameters

9.19.2.8 hpss_PIOSetOffset()

```
int hpss_PIOSetOffset (
    hpss_pio_t * Ctx,
    uint64_t Offset )
```

Set the offset in the PIO context.

Parameters

in, out	<i>Ctx</i>	Context to modify
in	<i>Offset</i>	HPSS offset to begin operating on

Update the offset for the context to the provided value. The next transfer will begin at the provided offset.

Return values

0	Success
-EINVAL	Invalid input parameters

9.19.2.9 hpss_PIOSetStripeWidth()

```
int hpss_PIOSetStripeWidth (
    hpss_pio_t * Ctx,
    uint32_t Width )
```

Set the client stripe width.

Parameters

<i>in, out</i>	<i>Ctx</i>	Context to modify
<i>in</i>	<i>Width</i>	Client stripe width

Update the stripe width for the context to the provided value.

Return values

<i>0</i>	Success
<i>-EINVAL</i>	Invalid input parameters

9.19.2.10 `hpss_PIOSetUseAPIHost()`

```
int hpss_PIOSetUseAPIHost (
    hpss_pio_t * Ctx,
    int UseAPIHost )
```

Set the client data hostname behavior.

Parameters

<i>in, out</i>	<i>Ctx</i>	Context to modify
<i>in</i>	<i>UseAPIHost</i>	Use API Host (true/false)

Update the API host

Return values

<i>0</i>	Success
<i>-EINVAL</i>	Invalid input parameters

9.19.2.11 `hpss_PIOWriteFile()`

```
int hpss_PIOWriteFile (
    int hpss_fd,
    uint64_t data_length,
    int local_fd,
    uint64_t * bytes_copied )
```

Write a file to HPSS from a local file using PIO.

Parameters

in	<i>hpss_fd</i>	Open HPSS File Descriptor
in	<i>data_length</i>	Amount of data to write to HPSS
in	<i>local_fd</i>	Open local file descriptor to read from
out	<i>bytes_copied</i>	Number of bytes copied to <i>hpss_fd</i>

This function will use PIO with default settings (from `hpss_PIOSetDefaultCtx`) to copy data from the *local_fd* to *hpss_fd*, up to *data_length* bytes.

Return values

0	Success
-EIO	I/O Error
-EINVAL	Invalid input parameters

9.19.2.12 `hpss_PIOWriteFileMulti()`

```
int hpss_PIOWriteFileMulti (
    hpss_pio_t * Ctx,
    int hpss_fd,
    uint64_t data_length,
    int local_fd,
    uint64_t * bytes_copied )
```

Write a file to HPSS from a local file using PIO.

Parameters

in, out	<i>Ctx</i>	Context to use
in	<i>hpss_fd</i>	Open HPSS File Descriptor
in	<i>data_length</i>	Amount of data to write to HPSS
in	<i>local_fd</i>	Open local file descriptor to read from
out	<i>bytes_copied</i>	Number of bytes copied to <i>hpss_fd</i>

This function will use PIO with default settings (from `hpss_PIOSetDefaultCtx`) to copy data from the *local_fd* to *hpss_fd*, up to *data_length* bytes.

Return values

0	Success
-EIO	I/O Error
-EINVAL	Invalid input parameters

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, unsigned32 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 hpss_GetTrashcan()

```
int hpss_GetTrashcan (
    const char * Path,
    char * TrashcanBuf,
    int TrashcanBufLength,
    ns_ObjHandle_t * FilesetRoot )
```

Locate the trashcan for a provided path.

Parameters

in	<i>Path</i>	Path to lookup trashcan for
in, out	<i>TrashcanBuf</i>	Buffer for trashcan path
in	<i>TrashcanBufLength</i>	TrashcanBuf's length
in, out	<i>FilesetRoot</i>	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

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EFAULT</i>	Path or Trashcan Buffer not provided.
<i>HPSS_ENOENT</i>	Path does not exist
<i>HPSS_ENOTDIR</i>	Some component of the path is not a directory
<i>HPSS_ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit
<i>HPSS_EINVAL</i>	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 hpss_GetTrashcanHandle()

```
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	<i>ObjHandle</i>	Object Handle for lookup
in	<i>Path</i>	Path to lookup trashcan for
in	<i>Ucred</i>	User Credentials
in, out	<i>TrashcanBuf</i>	Buffer for trashcan path
in	<i>TrashcanBufLength</i>	TrashcanBuf's length
in, out	<i>FilesetRoot</i>	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

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_ENAMETOOLONG</i>	The length of the Path argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
<i>HPSS_EFAULT</i>	Path or Trashcan Buffer not provided.
<i>HPSS_ENOENT</i>	Path does not exist
<i>HPSS_ENOTDIR</i>	Some component of the path is not a directory
<i>HPSS_EINVAL</i>	Invalid trashcan length specified or no handle provided.

Note

See `hpss_GetTrashcan` for additional notes about this function.

9.20.2.3 `hpss_GetTrashSettings()`

```
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	<i>TrashSettings</i>	Trashcan Settings

Return values

<i>0</i>	No error. Trash settings in the buffer.
<i>-EFAULT</i>	The TrashSettings was null.
<i>-EACCES</i>	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 `hpss_ReadTrash()`

```
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	<i>SubsystemId</i>	directory object handle
in	<i>UserIdP</i>	user id - if NULL, the authenticated user's credentials are used
in	<i>RealmIdP</i>	realm id - if NULL, the local realm is used
in	<i>BufferSize</i>	size of output buffer (in bytes)
out	<i>End</i>	hit end of trash
out	<i>TrashPtr</i>	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 hpss_Undelete()

```

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	<i>Path</i>	Path of file to undelete
in	<i>Flag</i>	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 hpss_UndeleteHandle()

```
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	<i>ObjHandle</i>	Parent object handle
in	<i>Path</i>	Path of file to undelete
in	<i>Ucred</i>	User credentials
in	<i>Flag</i>	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 Data Containers

Functions which implement data containers.

Files

- file [hpss_ihashtable.h](#)
Generic intrusive hash table.
- file [hpss_ilst.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.
- struct [hpss_irbtree_node](#)
A node in the tree.

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\_ilst\_t (*)(hpss\_ilst\_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\_ilst\_destructor\_t) (hpss\_ilst\_node\_t *Node)`
Node destructor.
- `typedef struct hpss\_ilst\_iterator\_token * hpss\_ilst\_iterator\_token\_h`
Typedef for opaque token.
- `typedef hpss\_ilst\_foreach\_retval\_e (* hpss\_ilst\_mapped\_func\_t) (hpss\_ilst\_node\_t *Node)`
Mapping function type.
- `typedef struct hpss\_ilst\_node hpss\_ilst\_node\_t`
Just the typedef for the following [hpss_ilst_node](#) struct.
- `typedef struct hpss\_ilst hpss\_ilst\_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_ilst_append (hpss_ilst_t *List, hpss_ilst_node_t *Node)`
Add a node to the end of an existing list.
- `void hpss_ilst_clear (hpss_ilst_t *List)`
Clear an entire list by taking out all nodes and freeing memory.
- `hpss_ilst_t * hpss_ilst_custom_init (hpss_ilst_destructor_t Destroy, hpss_ilst_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_ilst_empty (hpss_ilst_t *List)`
Check whether a list node points to anything else.
- `void hpss_ilst_foreach (hpss_ilst_t *List, hpss_ilst_mapped_func_t F)`
Map a function across all nodes in a list.
- `void hpss_ilst_foreach_range (hpss_ilst_t *List, hpss_ilst_mapped_func_t F, int32_t StartIndex, int32_t EndIndex)`
Map a function across all nodes in a given range of a list.
- `hpss_ilst_node_t * hpss_ilst_get_current (const hpss_ilst_t *List)`
Return the current node in the list, useful for starting iteration.
- `hpss_ilst_node_t * hpss_ilst_get_current_r (const hpss_ilst_iterator_token_h Token)`
Return the current node in the list, useful for starting reentrant iteration.
- `hpss_ilst_node_t * hpss_ilst_get_node (hpss_ilst_t *List, int32_t Index)`
Get a node at a given index in a list.
- `int32_t hpss_ilst_get_size (const hpss_ilst_t *const List)`
Access the size variable in the private hpss_ilst_t struct.
- `void hpss_ilst_halt_iterator (hpss_ilst_t *List)`
Halt iteration, reset Start, Current, and End to NULL.
- `hpss_ilst_t * hpss_ilst_init (hpss_ilst_destructor_t Destroy)`
Initialize a list.
- `int32_t hpss_ilst_init_iterator (hpss_ilst_t *List)`
Init the iterator to iterate over the entire list.
- `hpss_ilst_iterator_token_h hpss_ilst_init_iterator_r (hpss_ilst_t *List)`
Define a range of indices to include during iteration. Also initialize token for reentrant iteration.
- `int32_t hpss_ilst_init_iterator_range (hpss_ilst_t *List, int32_t Start, int32_t End)`
Init the Current, Start, and End variables in the hpss_ilst struct by giving a range of indices to iterate over.
- `hpss_ilst_iterator_token_h hpss_ilst_init_iterator_range_r (hpss_ilst_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_ilst_insert (hpss_ilst_t *List, hpss_ilst_node_t *Node1, hpss_ilst_node_t *Node2)`
Insert a node into an existing list.
- `int32_t hpss_ilst_join (hpss_ilst_t *List1, hpss_ilst_t *List2)`
Attach one list to the end of another list.
- `int32_t hpss_ilst_prepend (hpss_ilst_t *List, hpss_ilst_node_t *Node)`
Add a node to the start of an existing list.
- `int32_t hpss_ilst_reinit_iterator (hpss_ilst_t *List)`
Reinitialize iteration on the entire list using the halt and initialize functions above.
- `int32_t hpss_ilst_reinit_iterator_range (hpss_ilst_t *List, int32_t Start, int32_t End)`
Reinitialize iteration on a range of indices using the halt and initialize functions above.
- `void hpss_ilst_remove (hpss_ilst_t *List, hpss_ilst_node_t *Node)`
Remove a list node from a list.
- `void hpss_ilst_remove_safe (hpss_ilst_t *List, hpss_ilst_node_t *Node)`
Fail-safe function to 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 *REnd, 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.

9.21.1 Detailed Description

Functions which implement data containers.

9.21.2 Class Documentation

9.21.2.1 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

9.21.2.2 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

9.21.3 Macro Definition Documentation

9.21.3.1 hpss_irbtree_container

```
#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	<i>ptr</i>	Pointer to <code>bfs_TreeNode_t</code>
in	<i>type</i>	Type of the struct containing the <code>bfs_TreeNode_t</code>
in	<i>member</i>	Member of <i>type</i> that <i>ptr</i> points to

Note

You should only use this when you absolutely know what struct this tree node belongs to.

9.21.4 Typedef Documentation

9.21.4.1 `hpss_ilst_alloc_func_t`

```
typedef hpss\_ilst\_t*(hpss\_ilst\_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_ilst_custom_init\(\)](#) so calling code can specify a method of dynamically allocating the memory for the `hpss_ilst_t`

See also

[hpss_ilst_custom_init\(\)](#)

9.21.4.2 `hpss_ilst_destructor_t`

```
typedef void(* hpss\_ilst\_destructor\_t) (hpss\_ilst\_node\_t *Node)
```

Node destructor.

Use this as a callback to cleanup the struct containing a node

See also

[hpss_ilst_clear](#)

9.21.4.3 `hpss_ilst_iterator_token_h`

```
typedef struct hpss\_ilst\_iterator\_token* hpss\_ilst\_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

9.21.4.4 `hpss_ilst_mapped_func_t`

```
typedef hpss\_ilst\_foreach\_retval\_e(* hpss_ilst_mapped_func_t) (hpss\_ilst\_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_ilst_foreach\(\)](#) function

Note

To declare a function of this type, you need not use the `hpss_ilst_mapped_func_t` keyword, just make a function that takes an `hpss_ilst_node_t*` node and returns `KEEP_ITERATING` or `STOP_ITERATING`

See also

[hpss_ilst_foreach](#)

9.21.4.5 `hpss_ilst_t`

```
typedef struct hpss_ilst hpss\_ilst\_t
```

List type.

List object that acts as a handle for a list to be passed to `hpss_ilst` 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

9.21.4.6 `hpss_irbtree_node_callback_t`

```
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](#)

9.21.4.7 `hpss_irbtree_node_comparator_t`

```
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](#)

9.21.4.8 `hpss_irbtree_node_destructor_t`

```
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](#)

9.21.4.9 `hpss_irbtree_node_printer_t`

```
typedef void(* hpss_irbtree_node_printer_t) (const hpss\_irbtree\_node\_t *Node)
```

Callback to print a node.

See also

[hpss_irbtree_print](#)

9.21.5 Enumeration Type Documentation

9.21.5.1 `hpss_ihashtable_constants`

```
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.

9.21.5.2 hpss_irbtree_color

```
enum hpss_irbtree_color
```

Node colors.

Enumerator

IRBTREE_RED	The node is colored red
IRBTREE_BLACK	The node is colored black

9.21.6 Function Documentation

9.21.6.1 hpss_ilst_append()

```
int32_t hpss_ilst_append (
    hpss_ilst_t * List,
    hpss_ilst_node_t * Node )
```

Add a node to the end of an existing list.

This is useful for implementing queues

Parameters

in	<i>List</i>	List to append to
in	<i>Node</i>	Node to append

Returns

-1 if the insertion operation did not succeed, 0 otherwise

9.21.6.2 hpss_ilst_clear()

```
void hpss_ilst_clear (
    hpss_ilst_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	<i>List</i>	List to destroy
in	<i>Destructor</i>	Callback to use on each node

9.21.6.3 hpss_ilst_custom_init()

```
hpss_ilst_t* hpss_ilst_custom_init (
    hpss_ilst_destructor_t Destroy,
    hpss_ilst_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_ilst_t`. Similar to `hpss_ilst_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	<i>Destroy</i>	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 <code>hpss_ilst_remove</code> in order to take a node out of a list but not destroy it so it can still be used later.
in	<i>AllocFunc</i>	Allocator function that defines how to dynamically allocate the memory for the <code>hpss_ilst_t</code> struct. If the calling code does not require a specific allocator, use the generic <code>hpss_ilst_init()</code> function instead.
in	<i>RequestID</i>	A request ID of type <code>hpss_reqid_t</code> for logging purposes
in	<i>FunctionName</i>	The name of the function that called this init function, also for logging purposes
in	<i>FileName</i>	The name of the file from which <code>hpss_ilst_custom_init()</code> was called, also for logging purposes
in	<i>LineNumber</i>	The line number where <code>hpss_ilst_custom_init()</code> was called, also for logging purpose

9.21.6.4 hpss_ilst_empty()

```
bool hpss_ilst_empty (
    hpss_ilst_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_ilst_empty` will return True.

Parameters

in	List	List to check

Returns

True if List is empty, false if List is not empty

9.21.6.5 hpss_ilst_foreach()

```
void hpss_ilst_foreach (
    hpss_ilst_t * List,
    hpss_ilst_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 <code>hpss_ilst_mapped_func_t</code> , takes a pointer to an <code>hpss_ilst_node_t</code> 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

9.21.6.6 hpss_ilst_foreach_range()

```
void hpss_ilst_foreach_range (
    hpss_ilst_t * List,
    hpss_ilst_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	<i>List</i>	A pointer to the <code>hpss_ilst_t</code> we will be mapping a function over
in	<i>F</i>	A function of type <code>hpss_ilst_mapped_func_t</code> , takes a pointer to an <code>hpss_ilst_node_t</code> as its only argument
in	<i>StartIndex</i>	Which node to start with
in	<i>EndIndex</i>	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_ilst_foreach_range(List, func, 0, size_of_list-1)` is the same as `hpss_ilst_foreach(List, func)`

9.21.6.7 hpss_ilst_get_current()

```
hpss_ilst_node_t* hpss_ilst_get_current (
    const hpss_ilst_t * List )
```

Return the current node in the list, useful for starting iteration.

Parameters

in	<i>List</i>	Pointer to a list structure

Returns

The current node stored in `List->Current`, NULL if the List is empty

9.21.6.8 hpss_ilst_get_current_r()

```
hpss_ilst_node_t* hpss_ilst_get_current_r (
    const hpss_ilst_iterator_token_h Token )
```

Return the current node in the list, useful for starting reentrant iteration.

Parameters

in	<i>Token</i>	Token that has state information for reentrant iteration

Returns

The current node stored in *Token->Current*, NULL if the List is empty

9.21.6.9 hpss_ilst_get_node()

```
hpss_ilst_node_t* hpss_ilst_get_node (
    hpss_ilst_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	<i>List</i>	Pointer to a list structure
in	<i>Index</i>	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*

9.21.6.10 hpss_ilst_get_size()

```
int32_t hpss_ilst_get_size (
    const hpss_ilst_t *const List )
```

Access the size variable in the private *hpss_ilst_t* struct.

Parameters

in	<i>List</i>	an <i>hpss_ilst_t</i> whose size we want to return

Returns

size of the list as a 4 byte integer

9.21.6.11 hpss_ilst_halt_iterator()

```
void hpss_ilst_halt_iterator (
    hpss_ilst_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_ilst_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_ilst_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_ilst_halt_iterator\(\)](#) is called.

Parameters

in	<i>List</i>	A List pointer of type <code>hpss_ilst_t*</code>

9.21.6.12 hpss_ilst_init()

```
hpss_ilst_t* hpss_ilst_init (
    hpss_ilst_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	<i>Destroy</i>	a destructor of type <code>hpss_ilst_destructor_t*</code>

Returns

List List that was initialized

9.21.6.13 hpss_ilst_init_iterator()

```
int32_t hpss_ilst_init_iterator (
    hpss_ilst_t * List )
```

Init the iterator to iterate over the entire list.

Parameters

<code>in</code>	<code>List</code>	A List pointer of type <code>hpss_ilst_t*</code>

Returns

0 for success, `HPSS_E_EXIST` for failure

9.21.6.14 `hpss_ilst_init_iterator_r()`

```
hpss_ilst_iterator_token_h hpss_ilst_init_iterator_r (
    hpss_ilst_t * List )
```

Define a range of indices to include during iteration. Also initialize token for reentrant iteration.

Parameters

<code>in</code>	<code>List</code>	A List pointer of type <code>hpss_ilst_t*</code>

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

9.21.6.15 `hpss_ilst_init_iterator_range()`

```
int32_t hpss_ilst_init_iterator_range (
    hpss_ilst_t * List,
    int32_t Start,
    int32_t End )
```

Init the Current, Start, and End variables in the `hpss_ilst` 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	<i>List</i>	A List pointer of type <code>hpss_ilst_t*</code>
in	<i>Start</i>	Index of first node in the list we will iterate over
in	<i>End</i>	Index of last node to be included in iteration

Returns

0 for success, `HPSS_E_EXIST` for failure

9.21.6.16 `hpss_ilst_init_iterator_range_r()`

```
hpss_ilst_iterator_token_h hpss_ilst_init_iterator_range_r (
    hpss_ilst_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	<i>List</i>	A List pointer of type <code>hpss_ilst_t*</code>
in	<i>Start</i>	First node to in the sublist we will iterate over
in	<i>End</i>	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

9.21.6.17 `hpss_ilst_insert()`

```
int32_t hpss_ilst_insert (
    hpss_ilst_t * List,
    hpss_ilst_node_t * Node1,
    hpss_ilst_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	<i>List</i>	List to insert node into
in	<i>Node1</i>	Node that is right before where we want Node2 to go
in	<i>Node2</i>	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

9.21.6.18 hpss_ilst_join()

```
int32_t hpss_ilst_join (
    hpss_ilst_t * List1,
    hpss_ilst_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	<i>List1</i>	List pointer, this will be the list that becomes longer
in	<i>List2</i>	List pointer, this list will become empty and get reinitialized

Returns

-1 if the lists could not be joined, 0 otherwise

9.21.6.19 hpss_ilst_prepend()

```
int32_t hpss_ilst_prepend (
    hpss_ilst_t * List,
    hpss_ilst_node_t * Node )
```

Add a node to the start of an existing list.

This is useful for implementing stacks

Parameters

in	<i>List</i>	List to prepend to
in	<i>Node</i>	Node to prepend

Returns

-1 if the insertion operation did not succeed, 0 otherwise

9.21.6.20 `hpss_ilst_reinit_iterator()`

```
int32_t hpss_ilst_reinit_iterator (
    hpss_ilst_t * List )
```

Reinitialize iteration on the entire list using the halt and initialize functions above.

This function is a wrapper function that calls `hpss_ilst_init_iterator()`, logs any error that came from the init, then calls `hpss_ilst_halt_iterator()` and `hpss_ilst_init_iterator()` if the first attempt at initializing returned a nonzero error code.

Parameters

in	<i>List</i>	A List pointer of type <code>hpss_ilst_t*</code>

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

9.21.6.21 `hpss_ilst_reinit_iterator_range()`

```
int32_t hpss_ilst_reinit_iterator_range (
    hpss_ilst_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_ilst_init_iterator()`, logs any error that came from the init, then calls `hpss_ilst_halt_iterator()` and `hpss_ilst_init_iterator()` if the first attempt at initializing returned a nonzero error code.

Parameters

in	<i>List</i>	A List pointer of type <code>hpss_ilst_t*</code>
in	<i>Start</i>	The index of the first node to include in the iteration
in	<i>End</i>	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

9.21.6.22 hpss_ilst_remove()

```
void hpss_ilst_remove (
    hpss_ilst_t * List,
    hpss_ilst_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	<i>List</i>	List pointer
in	<i>Node</i>	Node to remove

9.21.6.23 hpss_ilst_remove_safe()

```
void hpss_ilst_remove_safe (
    hpss_ilst_t * List,
    hpss_ilst_node_t * Node )
```

Fail-safe function to remove a list node from a list,.

This will remove a single node from a list, joining its two ends together. Unlike hpss_ilst_remove, it will not crash if the node is already removed or the list is empty.

Parameters

in	<i>List</i>	List pointer
in	<i>Node</i>	Node to remove

Note

Unlike hpss_ilst_remove, this function will not crash if Node is not in List. However, the non-safe remove operation is O(1) and this operation is O(n) (due to the verification). Consider whether performance is critical or if lists are likely to grow to be extremely large before using.

9.21.6.24 hpss_irbtree_clear()

```
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	<i>Tree</i>	Tree to destroy
in	<i>Destructor</i>	Callback to use on every node in the tree

9.21.6.25 hpss_irbtree_empty()

```
int hpss_irbtree_empty (
    const hpss_irbtree_t * Tree )
```

Check if a tree is empty.

Parameters

in	<i>Tree</i>	Tree to check for emptiness

Returns

whether the tree is empty

9.21.6.26 hpss_irbtree_find()

```
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	<i>Tree</i>	Tree to search
in	<i>Node</i>	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);
}
```

9.21.6.27 hpss_irbtree_find_cmp()

```
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	<i>Tree</i>	Tree to search
in	<i>Node</i>	Node to compare
in	<i>Comparator</i>	Alternate comparator
in	<i>KeyCompare</i>	Whether <i>Comparator</i> 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.

9.21.6.28 hpss_irbtree_foreach()

```
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	<i>Tree</i>	Tree to print
in	<i>Callback</i>	Function to call on each node
in	<i>CallbackArg</i>	Argument to pass to callback

Returns

return code from last callback
0 if the tree is empty

9.21.6.29 hpss_irbtree_foreach_node()

```
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	<i>Begin</i>	Node at the beginning of the range (inclusive)
in	<i>End</i>	Node at the end of the range (exclusive)
in	<i>Callback</i>	Function to call on each node
in	<i>CallbackArg</i>	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.

9.21.6.30 hpss_irbtree_foreach_node_prev()

```
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	<i>RBegin</i>	Node at the end of the range (inclusive)
in	<i>REnd</i>	Node at the beginning of the range (exclusive)
in	<i>Callback</i>	Function to call on each node
in	<i>CallbackArg</i>	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.

9.21.6.31 hpss_irbtree_init()

```
void hpss_irbtree_init (
    hpss_irbtree_t * Tree,
    hpss_irbtree_node_comparator_t Comparator )
```

Initialize a tree.

Parameters

in	<i>Tree</i>	Tree to initialize
in	<i>Comparator</i>	Comparator to use for tree

9.21.6.32 hpss_irbtree_insert()

```
hpss_irbtree_node_t* hpss_irbtree_insert (
    hpss_irbtree_t * Tree,
    hpss_irbtree_node_t * Node )
```

Insert a node into a tree.

Parameters

in	<i>Tree</i>	Tree to insert into
in	<i>Node</i>	Node to insert

Returns

existing node if duplicate found

inserted node if no duplicate found

9.21.6.33 hpss_irbtree_insert_multi()

```
void hpss_irbtree_insert_multi (
    hpss_irbtree_t * Tree,
    hpss_irbtree_node_t * Node )
```

Insert a (possibly) duplicate node into a tree.

Parameters

in	<i>Tree</i>	Tree to insert into
in	<i>Node</i>	Node to insert

9.21.6.34 hpss_irbtree_lower_bound()

```
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	<i>Tree</i>	Tree to search
in	<i>Node</i>	Node to compare

Returns

lower-bound node from tree
NULL for no match

9.21.6.35 hpss_irbtree_max()

```
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	<i>Tree</i>	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.

9.21.6.36 hpss_irbtree_min()

```
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	<i>Tree</i>	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.

9.21.6.37 hpss_irbtree_node_next()

```
hpss_irbtree_node_t* hpss_irbtree_node_next (  
    const hpss_irbtree_node_t * Node )
```

Iterate to the next node in the tree.

Parameters

in	<i>Node</i>	Current node

Returns

next node in the tree
NULL for the last node in the tree

9.21.6.38 hpss_irbtree_node_prev()

```
hpss_irbtree_node_t* hpss_irbtree_node_prev (  
    const hpss_irbtree_node_t * Node )
```

Iterate to the previous node in the tree.

Parameters

in	<i>Node</i>	Current node

Returns

previous node in the tree
NULL for the first node in the tree

9.21.6.39 hpss_irbtree_print()

```
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	<i>Tree</i>	Tree to print
in	<i>Printer</i>	Function to call on each node

9.21.6.40 hpss_irbtree_remove()

```
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	<i>Tree</i>	Tree to remove node from
in	<i>Node</i>	Node to remove
in	<i>Destructor</i>	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;
}
```

9.21.6.41 hpss_irbtree_size()

```
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

9.21.6.42 hpss_irbtree_upper_bound()

```
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

9.22 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.

9.22.1 Detailed Description

Prototypes for HPSS Environment variable functions.

9.22.2 Function Documentation

9.22.2.1 hpss_Envcpy()

```
signed32 hpss_Envcpy (
    char * Buf,
    char const * Env,
    unsigned32 Bufsize )
```

Copies an environment value into a user-supplied buffer.

Parameters

in	<i>Buf</i>	User supplied string buffer
in	<i>Env</i>	Name of the environment variable
in	<i>Bufsize</i>	Size of <i>Buf</i>

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

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_ENOMEM</i>	<i>Buf</i> NULL or 0 Length
<i>HPSS_ENAMETOOLONG</i>	<i>Buf</i> too small
<i>HPSS_ENOENT</i>	<i>Env</i> is not defined

9.22.2.2 hpss_EnvLocalUpdate()

```
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

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EAGAIN</i>	Environment not initialized
<i>HPSS_ENOMEM</i>	Memory allocation error
<i>HPSS_EINVAL</i>	Invalid input argument

Note

This function changes only the environment local to the running process which called this function.

9.22.2.3 hpss_EnvReInitialize()

```
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

9.22.2.4 hpss_Getenv()

```
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.

9.23 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_universaladdressstoport](#) (const char *address)
Return the port from the universal address representation of the address structure.

9.23.1 Detailed Description

Functions for facilitating IPV4 and IPV6 Networking.

9.23.2 Function Documentation

9.23.2.1 hpss_net_accept()

```
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	<i>sockfd</i>	Listening socket
out	<i>addr</i>	Sockaddr of the client
out	<i>errbuf</i>	Buffer for more detailed error info
in	<i>errbuflen</i>	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

<i>Positive</i>	File descriptor of the accepted client socket
<i>-1</i>	Error; refer to <code>errno</code> for specific error code

9.23.2.2 `hpss_net_addrmatch()`

```
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	<i>Address</i>	First network address
in	<i>Mask</i>	Network mask
in	<i>Entry</i>	Second network address

Compares two network addresses, byte by byte. If *Mask* is NULL, then all address bits are significant.

Return values

<i>0</i>	No matching entry
<i>1</i>	Addresses match

9.23.2.3 `hpss_net_bind()`

```
int hpss_net_bind (
    int sockfd,
    hpss_sockaddr_t * addr,
```

```
char * errbuf,  
size_t errbuflen )
```

Bind an unnamed socket to an address.

Parameters

in	<i>sockfd</i>	Listening socket
out	<i>addr</i>	Sockaddr of the client
out	<i>errbuf</i>	Error buffer
in	<i>errbuflen</i>	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.

9.23.2.4 hpss_net_connect()

```
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	<i>sockfd</i>	Socket to be connected
in, out	<i>addr</i>	Sockaddr to connect to
out	<i>errbuf</i>	Buffer for more detailed error info
in	<i>errbuflen</i>	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.

9.23.2.5 hpss_net_generatemask()

```
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	<i>source</i>	Template address
out	<i>mask</i>	Generated netmask
in	<i>bits</i>	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.

9.23.2.6 hpss_net_get_family_option()

```
int hpss_net_get_family_option (
    char * errbuf,
    size_t errbuflen )
```

Retrieves the `HPSS_NET_FAMILY` environment variable.

Parameters

out	<i>errbuf</i>	Buffer for more detailed error info
in	<i>errbuflen</i>	Length of <i>errbuf</i>

[`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

<code>HPSS_NET_IPV6</code>	IPv6 / IPv4 Mixed Mode
<code>HPSS_NET_IPV6_ONLY</code>	IPv6 Only
<code>HPSS_NET_IPV4_ONLY</code>	IPv4 Only
<code>HPSS_NET_UNSPECIFIED</code>	Unspecified

9.23.2.7 `hpss_net_get_family_string()`

```
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.

9.23.2.8 `hpss_net_getinaddr()`

```
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	<i>addr</i>	Socket address to get the internal representation of the address from
out	<i>addrfamily</i>	Address family for addr (AF_INET or AF_INET6)
out	<i>addrlen</i>	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.

9.23.2.9 hpss_net_getnameinfo()

```
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	<i>addr</i>	Socket address
out	<i>host</i>	Hostname
in	<i>hostlen</i>	Size of the host buffer
out	<i>serv</i>	Service/Port
in	<i>servlen</i>	Size of the service buffer
in	<i>flags</i>	Flags to send getnameinfo
out	<i>errbuf</i>	Buffer for detailed error info
in	<i>errbuflen</i>	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

<i>0</i>	Success
<i>EAI_SYSTEM</i>	Check errno for more details
<i>Other</i>	EAI error code

Note

`getnameinfo()` returns non-POSIX error codes in general. In order to decode the error `gai_strerror()` should be used.

9.23.2.10 hpss_net_getpeername()

```
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	<i>sockfd</i>	Socket to get the remote peer from
out	<i>addr</i>	Socket address structure that will contain peer information
out	<i>errbuf</i>	Buffer for more detailed error info
in	<i>errbuflen</i>	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

9.23.2.11 hpss_net_getport()

```
int hpss_net_getport (
    const hpss_sockaddr_t * addr,
    char * errbuf,
    size_t errbuflen )
```

Return the port associated with a socket address.

Parameters

in	<i>addr</i>	Socket address to get port from
out	<i>errbuf</i>	Buffer for detailed error info
in	<i>errbuflen</i>	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

9.23.2.12 hpss_net_getuniversaladdress()

```
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	<i>addr</i>	Socket address structure to get the universal address from
out	<i>buf</i>	Buffer to store the universal address in
in	<i>buflen</i>	Length of buf
out	<i>errbuf</i>	Buf for detailed error info
in	<i>errbuflen</i>	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

<code>>0</code>	Length written to buf
<code><=0</code>	Negated POSIX error code

9.23.2.13 hpss_net_initaddr()

```
void hpss_net_initaddr (
    hpss_sockaddr_t * addr )
```

Initializes a `hpss_sockaddr_t`.

Parameters

out	<i>addr</i>	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`.

9.23.2.14 hpss_net_listen()

```
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	<i>sockfd</i>	Socket to perform the listen on
in	<i>backlog</i>	Size of the request queue
out	<i>errbuf</i>	Buffer for more detailed error info
in	<i>errbuflen</i>	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

9.23.2.15 hpss_net_setport()

```
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	<i>addr</i>	Sockaddr to store port
in	<i>port</i>	Port to store
out	<i>errbuf</i>	Buffer for more detailed error info
in	<i>errbuflen</i>	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

9.23.2.16 hpss_net_socket()

```
int hpss_net_socket (
    hpss_sockaddr_t * addr,
```

```

    int type,
    int protocol,
    char * errbuf,
    size_t errbuflen )

```

Open a new socket descriptor.

Parameters

in	<i>addr</i>	Sockaddr to get family from
in	<i>type</i>	Type of socket (TCP, UDP, etc.)
in	<i>protocol</i>	Protocol to use with socket
out	<i>errbuf</i>	Buf for detailed error info
in	<i>errbuflen</i>	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

9.23.2.17 hpss_net_strerror()

```

void hpss_net_strerror (
    int errnum,
    char * buf,
    size_t buflen )

```

Fill in the buffer with error string.

Parameters

in	<i>errnum</i>	Error code to use
out	<i>buf</i>	Buffer to be returned
in	<i>buflen</i>	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.

9.23.2.18 hpss_net_universaladdresstoport()

```
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

9.24 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.

9.24.1 Detailed Description

XML Interfaces.

9.24.2 Function Documentation

9.24.2.1 hpss_ChompXMLHeader()

```
char* hpss_ChompXMLHeader (
    char * XML,
    char * Header )
```

Remove the XML Header from an XML string, and optionally return it in its own string.

Parameters

<i>in</i>	<i>XML</i>	XML String
<i>in, out</i>	<i>Header</i>	Header component of XML

Return values

<i>NULL</i>	No header to remove, or resources unavailable
<i>Other</i>	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 original string.

9.24.2.2 hpss_CreateXMLWhere()

```
int hpss_CreateXMLWhere (
    hpss_userattr_list_t * Attrs,
    char * Query,
    int QueryLen )
```

Generates XMLExists statements.

Parameters

in	<i>Attrs</i>	Attribute List (only the first element is used)
out	<i>Query</i>	Query Buffer
in	<i>QueryLen</i>	Query Length

Return values

0	Success
-EINVAL	String exceeds supplied length

Note

Algorithm:

- Generate the XMLExists depending upon whether the Attr has a value

9.24.2.3 hpss_CreateXQueryDelete()

```
int hpss_CreateXQueryDelete (
    char * XPath,
    char * Query,
    int QueryLen )
```

Generates XQuery Delete statements.

Parameters

in	<i>XPath</i>	XPath to Generate Delete Query For
out	<i>Query</i>	Query Buffer
in	<i>QueryLen</i>	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

9.24.2.4 hpss_CreateXQueryGet()

```
int hpss_CreateXQueryGet (
    char * XPath,
    int XMLFlags,
    char * Query,
    int QueryLen )
```

Generates XQuery Get statements.

Parameters

in	<i>XPath</i>	XPath to Generate Retrieval Query For
in	<i>XMLFlags</i>	Flags for Retrieval Processing • UDA_API_XML retrieves raw XML • UDA_API_VALUE retrieves XML attribute value
out	<i>Query</i>	Query Buffer
in	<i>QueryLen</i>	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

9.24.2.5 hpss_CreateXQueryUpdate()

```
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	<i>XPath</i>	XPath to Generate Update Query For
in	<i>Value</i>	Value to set in Update Query
out	<i>Query</i>	Buffer for Generated Query
in	<i>QueryLen</i>	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

9.24.2.6 hpss_ValidXPath()

```
int hpss_ValidXPath (
    char * XPath )
```

Validates that the argument has some of the required components of an XPath statement.

Parameters

in	<i>XPath</i>	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 /.

9.24.2.7 hpss_XMLtoAttr()

```
int hpss_XMLtoAttr (
    char * XMLstr,
    hpss_userattr_list_t * Attrs,
    int Flags )
```

Convert an XML string into some number of Key/Value pairs.

Parameters

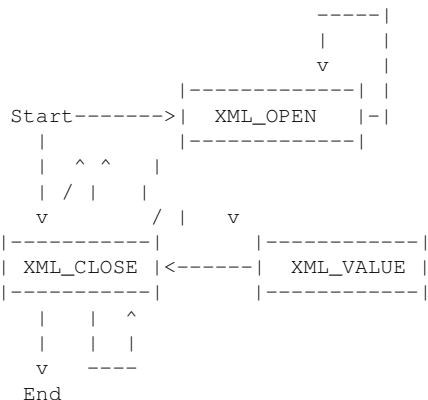
in	<i>XMLstr</i>	XML String to process
out	<i>Attrs</i>	User-defined Attributes List from XML
in	<i>Flags</i>	Flags to change processing behavior • XML_NO_ATTR will only handle XML Element Nodes • XML_ATTR will also handle XML Element Attributes

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



9.24.2.8 hpss_XPathBranch()

```
int hpss_XPathBranch (
    char * Path,
    char * Ret )
```

Obtain the XPath without the leaf component.

Parameters

<i>in</i>	<i>Path</i>	Full XPath
<i>in, out</i>	<i>Ret</i>	All but last component of Path

Return values

<i>-EFAULT</i>	An argument was NULL
<i>-EINVAL</i>	The supplied XPath is invalid
<i>-ENOENT</i>	There is no branch component
<i>0</i>	Success

Note

Algorithm:

- If Path or Ret is invalid Return error
- Find last occurrence of '/' Copy from first '/' to last '/'
- If not found return ENOENT

9.24.2.9 hpss_XPathLeaf()

```
int hpss_XPathLeaf (
    char * Path,
    char * Ret )
```

Obtain the last component of an XPath.

Parameters

<i>in</i>	<i>Path</i>	Full XPath
<i>in, out</i>	<i>Ret</i>	Last component of Path

Return values

<i>-EFAULT</i>	An argument was NULL
<i>-EINVAL</i>	The supplied XPath is invalid
<i>-ENOENT</i>	There is no leaf component
<i>-ERANGE</i>	XPath component is too large
<i>0</i>	Success

Note

Algorithm:

- If Path or Ret is invalid Return error
- Find last occurrence of '/' Copy from last '/' to end
- If not found return ENOENT

9.24.2.10 hpss_XPathRBranch()

```
int hpss_XPathRBranch (
    char * Path,
    char * Ret )
```

Obtain the XPath without the root component.

Parameters

<i>in</i>	<i>Path</i>	Full XPath
<i>in, out</i>	<i>Ret</i>	All but first component of Path

Return values

<i>-EFAULT</i>	An argument was NULL
<i>-EINVAL</i>	The supplied XPath is invalid
<i>-ENOENT</i>	There is no RBranch component
<i>0</i>	Success

Note

Algorithm:

- If Path or Ret is invalid Return error
- Find second occurrence of '/' Copy from second '/' to last '/'
- If not found return ENOENT

9.24.2.11 hpss_XPathRoot()

```
int hpss_XPathRoot (
    char * Path,
    char * Ret )
```

Obtain the first component of an XPath.

Parameters

<i>in</i>	<i>Path</i>	Full XPath
<i>in, out</i>	<i>Ret</i>	First Component of Path

Return values

<i>-EFAULT</i>	An argument was NULL
<i>-EINVAL</i>	The supplied XPath is invalid
<i>-ENOENT</i>	There is no first component
<i>0</i>	Success

Note

Algorithm:

- If Path or Ret is invalid Return error
- Find second occurrence of '/' Copy from first '/' to second
- If not found return ENOENT

9.24.2.12 hpss_XPathtoXML()

```
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

<i>in</i>	<i>Attr</i>	Attribute Name
<i>in</i>	<i>Val</i>	Attribute Value
<i>in, out</i>	<i>XMLstr</i>	XML String
<i>in</i>	<i>Len</i>	Length of XML String
<i>in</i>	<i>Insert</i>	Flag for modifying value expression when used for insertion.
<i>in</i>	<i>Level</i>	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

<i>0</i>	Success

Return values

-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

9.25 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)

9.25.1 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.

9.25.2 Macro Definition Documentation

9.25.2.1 add64_3m

```
#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.

9.25.2.2 add64m

```
#define add64m(  
    a1,  
    a2 )
```

Add two unsigned 64s, $a1$ and $a2$

9.25.2.3 and64m

```
#define and64m(  
    a1,  
    a2 )
```

Find the bitwise and of two 64-bit values, $a1$ and $a2$

9.25.2.4 andbit64m

```
#define andbit64m(  
    a,  
    b )
```

And bit position b in unsigned 64 bit integer a

9.25.2.5 bld64m

```
#define bld64m(  
    a,  
    b )
```

Build a unsigned64 from 32-bit integers a and b

9.25.2.6 cast32m

```
#define cast32m(  
    a )
```

Cast *a* to an unsigned 32 bit integer

9.25.2.7 cast64m

```
#define cast64m(  
    a )
```

Cast *a* to an unsigned 64

9.25.2.8 chkbit64m

```
#define chkbit64m(  
    a,  
    b )
```

Return 1 if bit position *b* is set in unsigned integer *a*

9.25.2.9 clrbit64m

```
#define clrbit64m(  
    a,  
    b )
```

Clear bit position *b* in unsigned 64 bit integer *a*

9.25.2.10 CONVERT_LONGLONG_TO_U64

```
#define CONVERT_LONGLONG_TO_U64(  
    LL,  
    U64 )
```

Convert a long long, *LL* to a u_signed64, *U64*

9.25.2.11 CONVERT_U64_TO_LONGLONG

```
#define CONVERT_U64_TO_LONGLONG(  
    U64,  
    LL )
```

Convert a u_signed64, *U64* to a long long, *LL*

9.25.2.12 dec64m

```
#define dec64m(  
    a1,  
    a2 )
```

Subtract *a2* from *a1* destroying old *a1*.

9.25.2.13 div2x64m

```
#define div2x64m(  
    a1,  
    a2 )
```

Divide two unsigned 64s, *a1* and *a2*

9.25.2.14 div64_3m

```
#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.

9.25.2.15 div64m

```
#define div64m(  
    a1,  
    a2 )
```

Divide a unsigned 64, *a1*, by a unsigned 32 *a2*

9.25.2.16 eq64m

```
#define eq64m(  
    a1,  
    a2 )
```

Returns TRUE if *a1* is equal to *a2*, or FALSE.

9.25.2.17 eqz64m

```
#define eqz64m(  
    a )
```

Returns TRUE if *a* equals zero, or FALSE.

9.25.2.18 ge64m

```
#define ge64m(  
    a1,  
    a2 )
```

Returns TRUE if *a1* is greater than or equal to *a2*, or FALSE.

9.25.2.19 gt64m

```
#define gt64m(  
    a1,  
    a2 )
```

Returns TRUE if *a1* is greater than *a2*, or FALSE.

9.25.2.20 high32m

```
#define high32m(  
    a )
```

Reference the upper 32 bits of unsigned64 *a*

9.25.2.21 inc64m

```
#define inc64m(  
    a1,  
    a2 )
```

Add *a2* to *a1* destroying old *a1*.

9.25.2.22 le64m

```
#define le64m(  
    a1,  
    a2 )
```

Returns TRUE if *a1* is less than or equal to *a2*, or FALSE.

9.25.2.23 low32m

```
#define low32m(  
    a )
```

Reference the lower 32 bits of unsigned64 *a*

9.25.2.24 lt64m

```
#define lt64m(  
    a1,  
    a2 )
```

Returns TRUE if *a1* is less than *a2*, or FALSE.

9.25.2.25 mem64m

```
#define mem64m(  
    a )
```

Build a 64-bit unsigned int from *a*, an array of unsigned characters

9.25.2.26 mod2x64m

```
#define mod2x64m(  
    a1,  
    a2 )
```

Modulus an unsigned 64, *a1* by an unsigned 64 *a2*

9.25.2.27 mod64_3m

```
#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.

9.25.2.28 mod64m

```
#define mod64m(  
    a1,  
    a2 )
```

Modulus an unsigned 64, *a1* by an unsigned 32 *a2*

9.25.2.29 mul64_3m

```
#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.

9.25.2.30 mul64m

```
#define mul64m(  
    a1,  
    a2 )
```

Multiply an unsigned 64, *a1* by an unsigned 32 *a2*

9.25.2.31 neq64m

```
#define neq64m(  
    a1,  
    a2 )
```

Returns TRUE if *a1* is not equal to *a2*, or FALSE.

9.25.2.32 neqz64m

```
#define neqz64m(  
    a )
```

Returns TRUE if *a* does not equal zero, or FALSE.

9.25.2.33 not64m

```
#define not64m(  
    a )
```

Find the bitwise not of a 64-bit value

9.25.2.34 or64m

```
#define or64m(  
    a1,  
    a2 )
```

Find the bitwise or of two 64-bit values, *a1* and *a2*

9.25.2.35 orbit64m

```
#define orbit64m(  
    a,  
    b )
```

Or bit position *b* in unsigned 64 bit integer *a*

9.25.2.36 shl64_3m

```
#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.

9.25.2.37 shl64_ipm

```
#define shl64_ipm(  
    a,  
    s )
```

Shift *a* left by *s* positions, destroying old *a*.

9.25.2.38 shl64m

```
#define shl64m(  
    a1,  
    a2 )
```

Shift an unsigned 64, *a1* left by an unsigned 32 *a2*

9.25.2.39 shr64_3m

```
#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.

9.25.2.40 shr64_ipm

```
#define shr64_ipm(  
    a,  
    s )
```

Shift *a* right by *s* positions, destroying old *a*.

9.25.2.41 shr64m

```
#define shr64m(  
    a1,  
    a2 )
```

Shift an unsigned 64, *a1* right by an unsigned 32 *a2*

9.25.2.42 sub64_3m

```
#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.

9.25.2.43 sub64m

```
#define sub64m(  
    a1,  
    a2 )
```

Subtract two unsigned 64s, *a1* and *a2*

9.26 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 [hpsscfx_ConfGetClientInterfaces](#) (const char *hostName, const char *serverName, int *nwlfCount, hpss_sockaddr_t **nwlfList, char ***nwlfNames)
Lookup client interfaces by host.
- [hpss_cfg_stanza_t * hpsscfx_ConfParse](#) (char cfgFile[])
Parse the configuration file.
- int [hpsscfx_ConfSetDirPaths](#) (char *theDirString)
Set the Directory Path(s) for the Configuration File.
- int [hpsscfx_ConfSetFileName](#) (char *theFilename)
Set the name of the config file.
- [hpss_cfg_stanza_t * hpsscfx_LookupHostName](#) (hpss_cfg_stanza_t *hostStanzaList, const char *hostName, int ignoreDomain)
Find a stanza for a hostname.
- int [hpsscfx_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 [hpsscfx_NetoptGetWriteSize](#) (int SocketDescriptor, hpss_sockaddr_t *IpAddr)
Return the network (TCP/IP) write size to be used for the specified connection.
- int [hpsscfx_NetoptSetSock](#) (int Fd, hpss_sockaddr_t *NetAddress)
Adjust the Socket characteristics.
- int [hpsscfx_PatternMatch](#) (char *theString, char *thePattern, int *patternError)
Pattern match function.

9.26.1 Detailed Description

Common header for HPSS Client API extensions.

9.26.2 Macro Definition Documentation

9.26.2.1 MIN_RESTRICTED_PORT

```
#define MIN_RESTRICTED_PORT 1024
```

Get the range of restricted ports that are allowed.

Parameters

out	<i>minPort</i>	If non-NULL, set on exit to minimum restricted port
out	<i>maxPort</i>	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`

9.26.3 Function Documentation

9.26.3.1 hpsscfx_ConfGetClientInterfaces()

```
int hpsscfx_ConfGetClientInterfaces (
    const char * hostName,
    const char * serverName,
    int * nwIfCount,
    hpss_sockaddr_t ** nwIfList,
    char *** nwIfNames )
```

Lookup client interfaces by host.

Parameters

in	<i>hostName</i>	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	<i>serverName</i>	NULL or server host to search for. If NULL, then the Default server host stanza is used, if found.
out	<i>nwIfCount</i>	Set on exit to count of interfaces
out	<i>nwIfList</i>	If non-NULL, set on exit to malloc'd list of local network addresses
out	<i>nwIfNames</i>	If non-NULL, set on exit to malloc'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 sub stanza
        }
    }

    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.

9.26.3.2 hpsscfx_ConfParse()

```

hpss_cfg_stanza_t* hpsscfx_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 [hpsscfx_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 hpsscfx_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.

9.26.3.3 hpsscfx_ConfSetDirPaths()

```
int hpsscfx_ConfSetDirPaths (
    char * theDirString )
```

Set the Directory Path(s) for the Configuration File.

Parameters

in	<i>theDirString</i>	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 `hpsscfx_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

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EFAULT</i>	NULL string supplied
<i>HPSS_ENOMEM</i>	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.

9.26.3.4 hpsscfx_ConfSetFileName()

```
int hpsscfx_ConfSetFileName (
    char * theFilename )
```

Set the name of the config file.

Parameters

in	<i>theFilename</i>	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 `hpsscfx_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

<code>HPSS_E_NOERROR</code>	Success
<code>HPSS_EFAULT</code>	NULL Filename supplied
<code>HPSS_ENOMEM</code>	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.

9.26.3.5 `hpsscfx_LookupHostName()`

```
hpss_cfg_stanza_t* hpsscfx_LookupHostName (
    hpss_cfg_stanza_t * hostStanzaList,
    const char * hostName,
    int ignoreDomain )
```

Find a stanza for a hostname.

Parameters

in	<code>hostStanzaList</code>	Head of host stanza list
in	<code>hostName</code>	Hostname to search for
in	<code>ignoreDomain</code>	Whether to ignore the domain portion of the hostname

This function attempts to find a stanza 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.

9.26.3.6 hpsscfx_NetoptFindEntry()

```
int hpsscfx_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	<i>NetAddr</i>	Network address to look up
out	<i>RetEntryPtr</i>	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

9.26.3.7 hpsscfx_NetoptGetWriteSize()

```
ssize_t hpsscfx_NetoptGetWriteSize (
    int SocketDescriptor,
    hpss_sockaddr_t * IpAddr )
```

Return the network (TCP/IP) write size to be used for the specified connection.

Parameters

in	<i>SocketDescriptor</i>	Connection file descriptor
in	<i>IpAddr</i>	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 `lpAddr` is 0 on entry, then it is determined by getting the peer address for the socket.

Returns

the write size to be used

9.26.3.8 hpsscfx_NetoptSetSock()

```
int hpsscfx_NetoptSetSock (
    int Fd,
    hpss_sockaddr_t * NetAddress )
```

Adjust the Socket characteristics.

Parameters

in	<i>Fd</i>	File descriptor
in	<i>NetAddress</i>	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

9.26.3.9 hpsscfx_PatternMatch()

```
int hpsscfx_PatternMatch (
    char * theString,
    char * thePattern,
    int * patternError )
```

Pattern match function.

Parameters

in	<i>theString</i>	String to test for a pattern match
in	<i>thePattern</i>	Pattern to compare against
out	<i>patternError</i>	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

<i>TRUE</i>	Pattern and string match
<i>FALSE</i>	Pattern and string do not match

9.27 HPSS Config File Template API

Template-based parser for config files.

Modules

- [Main](#)
- [Parsers](#)
- [Validators](#)

9.27.1 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.

9.28 Main

Functions

- void [hpsscfx_DisposeConf](#) (hpsscfx_ConfTemplate_t const *conf_template, void *conf_data)
Dispose of a conf_data structure.
- void [hpsscfx_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 [hpsscfx_ParseTemplate](#) (void *out, hpsscfx_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 * [hpsscfx_ReadConf](#) (char const *FileName, char *result, size_t result_sz)
Read a configuration file in.
- int [hpsscfx_ReadConfTemplate](#) (char const *file_name, hpsscfx_ConfTemplate_t const *conf_template, void *conf_data, char *result, size_t result_sz)
Read a configuration file into a conf_data structure.

9.28.1 Detailed Description

Main template parse functions

9.28.2 Function Documentation

9.28.2.1 hpsscfx_DisposeConf()

```
void hpsscfx_DisposeConf (
    hpsscfx_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	<i>conf_template</i>	The description of the configuration data to be freed
in	<i>conf_data</i>	The configuration data to free

9.28.2.2 hpsscfx_FormatMessage()

```
void hpsscfx_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	<i>entry</i>	The entry that resulted in an error
out	<i>result</i>	The buffer to fill with the message
in	<i>result_sz</i>	The size of the output buffer
in	<i>msg</i>	The format message and following field data values

9.28.2.3 hpsscfx_ParseTemplate()

```

int hpsscfx_ParseTemplate (
    void * out,
    hpsscfx_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	<i>out</i>	The output configuration structure
in	<i>tmpl</i>	The description of the configuration
in	<i>root</i>	The root of the parsed config file
in	<i>disposing</i>	True if freeing the configuration structure
out	<i>result</i>	The buffer to fill with an error message
in	<i>result_sz</i>	The size of the output buffer

Note

Does not affect the result string if no errors.

9.28.2.4 hpsscfx_ReadConf()

```

hpss_cfg_stanza_t* hpsscfx_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.

9.28.2.5 hpsscfx_ReadConfTemplate()

```
int hpsscfx_ReadConfTemplate (
    char const * file_name,
    hpsscfx_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	<i>file_name</i>	The name of the conf file to read in
in	<i>conf_template</i>	The description of the structure of the stanza and configuration data
out	<i>conf_data</i>	The structure which will hold the parsed stanza
out	<i>result</i>	A text message describing any errors encountered
in	<i>result_sz</i>	The size of the message buffer

9.29 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 const *name, 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 const *name, char *result, size_t result_sz)
Enumeration parser. All possible choices are offered in the config template using an array of [hpsscfx_ChoiceTemplate_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 const *name, 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 const *name, 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 const *name, 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 const *name, 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 const *name, 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 const *name, 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

9.29.1 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

9.29.2 Function Documentation

9.29.2.1 HPSSCFGX_ENUM()

```
int HPSSCFGX_ENUM (
    char const * input,
    void * out,
    void const * aux,
    hpss_cfg_stanza_t * entry,
    bool disposing,
    char const * name,
    char * result,
    size_t result_sz )
```

Enumeration parser. All possible choices are offered in the config template using an array of `hpsscfgx_ChoiceTemplate_t` structs as aux data.

Note

Note that comparisons with enum names are case insensitive.

9.29.2.2 HPSSCFGX_STRING()

```
int HPSSCFGX_STRING (
    char const * input,
    void * out,
    void const * aux,
    hpss_cfg_stanza_t * entry,
    bool disposing,
    char const * name,
    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.

9.30 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 const *name, 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 const *name, char *result, size_t result_sz)
 Declare this configuration parameter invalid.

9.30.1 Detailed Description

Special purpose parser functions. They are typically returned by the validator functions to indicate special behavior is required.

9.31 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.

9.31.1 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

9.32 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.

9.32.1 Detailed Description

Posix Compliant HPSS APIs (Unsupported)

9.32.2 Function Documentation

9.32.2.1 hpss_CreateWithHints()

```
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	<i>Path</i>	Path to file to be opened
in	<i>Mode</i>	Desired file permissions, if created
in	<i>HintsIn</i>	Desired class of service
in	<i>HintsPri</i>	Priorities of hints
out	<i>HintsOut</i>	Granted class of service

The 'hpss_CreateWithHints' function creates a file specified 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
<i>Other</i>	Negated POSIX error code

9.32.2.2 hpss_OpenWithHints()

```
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	<i>Path</i>	Path to file to be opened
in	<i>Oflag</i>	Type of file access • O_RDONLY • O_WRONLY • O_RDWR • O_APPEND • O_TRUNC
in	<i>Mode</i>	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	<i>HintsIn</i>	Desired class of service
in	<i>HintsPri</i>	Desired priority
out	<i>HintsOut</i>	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

9.32.2.3 hpss_PosixCreate()

```
int hpss_PosixCreate (
    char * Path,
    mode_t Mode )
```

Create an HPSS file.

Parameters

in	<i>Path</i>	Path to file to be opened
in	<i>Mode</i>	Desired file perms if create

The 'hpss_PosixCreate' function creates a file specified by *Path*, with permissions as specified by *Mode*

Return values

≥ 0	Open file handle
< 0	Negated POSIX error code

9.32.2.4 hpss_PosixFstat()

```
int hpss_PosixFstat (
    int Fildes,
    struct stat * Buf )
```

Retrieve information about an open file or directory.

Parameters

in	<i>Fildes</i>	Open file or directory to retrieve information for
out	<i>Buf</i>	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
<i>Other</i>	Negated POSIX error code

9.32.2.5 hpss_PosixLseek()

```
long hpss_PosixLseek (
    int Fildes,
    long Offset,
    int Whence )
```

Set the file offset for an open file handl.

Parameters

in	<i>Fildes</i>	File descriptor of open object
in	<i>Offset</i>	Number of bytes to calculate new offset
in	<i>Whence</i>	Origin for the seek

This function sets the file offset for the open file handle, 'Fildes'.

Parameters

in	<i>Fildes</i>	Open file handle for which the file offset is set
in	<i>Offset</i>	Number of bytes to be used in calculating the new file offset; dependent upon the value of <i>Whence</i> as to the final effect on the new file offset.
in	<i>Whence</i>	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

9.32.2.6 hpss_PosixLstat()

```
int hpss_PosixLstat (
    char * Path,
    struct stat * Buf )
```

Obtain information about a file or directory, without traversing symbolic links.

Parameters

in	<i>Path</i>	Path of file to retrieve information for
out	<i>Buf</i>	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

9.32.2.7 hpss_PosixOpen()

```
int hpss_PosixOpen (
    char * Path,
    int Oflag,
    mode_t Mode )
```

Open an HPSS file.

Parameters

in	<i>Path</i>	Path to file to be opened
in	<i>Oflag</i>	Type of file access
in	<i>Mode</i>	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_RDWR, 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.

9.32.2.8 hpss_PosixStat()

```
int hpss_PosixStat (
    char * Path,
    struct stat * Buf )
```

Obtain information about a file or directory.

Parameters

in	<i>Path</i>	Path of file to retrieve information for
out	<i>Buf</i>	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

9.33 Client API Location Services

Functions used by client applications to locate servers.

Functions

- `int32_t hpss_LocateRootCS (hpss_reqid_t RequestId, ls_map_t *Location)`
Find the local root Core Server.
- `int32_t hpss_LocateServerById (hpss_reqid_t RequestId, hpss_srvr_id_t ServerId, ls_map_t *Location)`
Locate a server by its Id This routine locates a server by its unique ID. If ServerId is zero (Invalid), information about the local root Core Server is returned.
- `int32_t hpss_LocateServersByType (hpss_reqid_t RequestId, hpss_id_t SiteId, uint32_t ServerType, uint32_t SubsystemId, ls_map_array_t **Locations)`
Locate matching servers for a site.
- `int32_t hpss_LocateSiteById (hpss_reqid_t RequestId, hpss_id_t SiteId, char *SiteName)`
Lookup a site's name by its id.
- `int32_t hpss_LocateSiteByName (hpss_reqid_t RequestId, char *SiteName, hpss_id_t *SiteId)`
Lookup a site's id by its name.

9.33.1 Detailed Description

Functions used by client applications to locate servers.

9.33.2 Function Documentation

9.33.2.1 hpss_LocateRootCS()

```
int32_t hpss_LocateRootCS (
    hpss_reqid_t RequestId,
    ls_map_t * Location )
```

Find the local root Core Server.

Parameters

in	<i>RequestId</i>	Id of the request to use for logging
out	<i>Location</i>	Returned map information about the root Core Server

Return values

0	Successful return
-EFAULT	Location is NULL
-ECONN	A communication error has occurred
-ENOENT	No information exists for the local root CS
-ENOMEM	Out of memory
-EMutex	An internal error occurred while accessing a mutex
HPSS_ESYSTEM	An internal error has occurred.
Otherwise	Some other error has occurred

9.33.2.2 hpss_LocateServerByID()

```
int32_t hpss_LocateServerByID (
    hpss_reqid_t RequestId,
    hpss_srvr_id_t ServerId,
    ls_map_t * Location )
```

Locate a server by its Id This routine locates a server by its unique ID. If ServerId is zero (Invalid), information about the local root Core Server is returned.

Parameters

in	<i>RequestId</i>	Id of this request to use when logging
in	<i>ServerId</i>	Id of the server to look up. If this is empty, information about the local root CS is returned.
out	<i>Location</i>	Returned map information about the server

Return values

0	Successful return
HPSS_EFAULT	Location is NULL
HPSS_ECONN	A communication error has occurred
HPSS_ENOENT	No information exists for the specified server
HPSS_ENOMEM	Out of memory
HPSS_EMutex	An internal error occurred while accessing a mutex
HPSS_ESYSTEM	An internal error has occurred.
Otherwise	Some other error has occurred

9.33.2.3 hpss_LocateServersByType()

```
int32_t hpss_LocateServersByType (
    hpss_reqid_t RequestId,
    hpss_id_t SiteId,
    uint32_t ServerType,
    uint32_t SubsystemId,
    ls_map_array_t ** Locations )
```

Locate matching servers for a site.

This routine returns information about all servers that match the criteria specified for a single site.

Parameters

in	<i>RequestId</i>	Id of this request to use when logging
in	<i>SiteId</i>	Id of the site to look up information for. If this is empty, the local site is assumed.
in	<i>ServerType</i>	Type of server to lookup (i.e. CORE_SERVER_TYPE). If this is zero, all servers are eligible.
in	<i>SubsystemId</i>	Subsystem identifier of the server. If this is zero, all servers are eligible.
in	<i>Locations</i>	Returned information array for matching servers

Return values

0	Successful return
HPSS_EFAULT	Locations is NULL
HPSS_ECONN	A communication error has occurred
HPSS_ENOENT	No server information matches criteria
HPSS_ENOMEM	Out of memory
HPSS_EMutex	An internal error occurred while accessing a mutex
HPSS_ESYSTEM	An internal error has occurred
Otherwise	Some other error has occurred

9.33.2.4 hpss_LocateSiteById()

```
int32_t hpss_LocateSiteById (
    hpss_reqid_t RequestId,
    hpss_id_t SiteId,
    char * SiteName )
```

Lookup a site's name by its id.

Parameters

in	<i>RequestId</i>	Id of this request to use when logging
in	<i>SiteId</i>	Id of the site to lookup information for. If this is NULL or points to a zero id then the local site is assumed.
out	<i>SiteName</i>	Returned name of the site. Max size is HPSS_MAX_SITE_NAME.

Return values

<i>0</i>	Successful return
<i>HPSS_EFAULT</i>	SiteName is NULL
<i>HPSS_ECONN</i>	A communication error has occurred
<i>HPSS_ENOENT</i>	No site information matches criteria
<i>HPSS_ENOMEM</i>	Out of memory
<i>HPSS_EMUTEX</i>	An internal error occurred while accessing a mutex
<i>HPSS_ESYSTEM</i>	An internal error has occurred
<i>Otherwise</i>	Some other error has occurred

9.33.2.5 hpss_LocateSiteByName()

```
int32_t hpss_LocateSiteByName (
    hpss_reqid_t RequestId,
    char * SiteName,
    hpss_id_t * SiteId )
```

Lookup a site's id by its name.

Parameters

in	<i>RequestId</i>	Id of this request to use when logging
in	<i>SiteName</i>	Name of the site to lookup. Max size is HPSS_MAX_SITE_NAME. If this is null the local site is assumed.
out	<i>SiteId</i>	Returned Id of the site found.

Return values

<i>0</i>	Successful return
<i>HPSS_EFAULT</i>	Siteld is NULL
<i>HPSS_ECONN</i>	A communication error has occurred
<i>HPSS_ENOENT</i>	No site information matches criteria
<i>HPSS_ENOMEM</i>	Out of memory
<i>HPSS_EMUTEX</i>	An internal error occurred while accessing a mutex
<i>HPSS_ESYSTEM</i>	An internal error has occurred
<i>Otherwise</i>	Some other error has occurred

9.34 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.

9.34.1 Detailed Description

Hashing and Checksumming Functions.

9.34.2 Macro Definition Documentation

9.34.2.1 HPSS_CKSUM_AFALG

```
#define HPSS_CKSUM_AFALG (0x00000004)
```

Support for af-alg

9.34.2.2 HPSS_CKSUM_SSE42

```
#define HPSS_CKSUM_SSE42 (0x00000002)
```

Support for sse42 (x86)

9.34.2.3 HPSS_CKSUM_VPMSUM

```
#define HPSS_CKSUM_VPMSUM (0x00000001)
```

Support for vpmsum (ppc)

9.34.3 Function Documentation

9.34.3.1 hpss_HashAppend()

```
int hpss_HashAppend (  
    hpss_hash_t Hash,  
    int Character )
```

Append a character to the hash stream.

Parameters

in	<i>Hash</i>	Hash Engine
in	<i>Character</i>	Character data to append

Return values

<i>0</i>	Success
<i>HPSS_EFAULT</i>	Invalid hash context
<i>HPSS_EINVAL</i>	Invalid hash type
<i>HPSS_ESYSTEM</i>	Hash engine system failure

9.34.3.2 hpss_HashAppendBuf()

```
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	<i>Hash</i>	Hash Engine
in	<i>Buffer</i>	Data Buffer
in	<i>Length</i>	Buffer length

Return values

<i>0</i>	Success
<i>HPSS_EFAULT</i>	Invalid hash context or buffer
<i>HPSS_EINVAL</i>	Invalid hash type
<i>HPSS_ESYSTEM</i>	Hash engine system failure

9.34.3.3 hpss_HashAppendStr()

```
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	<i>Hash</i>	Hash Engine
in	<i>String</i>	C String

Return values

0	Success
HPSS_EFAULT	Invalid hash context or string
HPSS_EINVAL	Invalid hash type
HPSS_ESYSTEM	Hash engine system failure

9.34.3.4 hpss_HashBaseInit()

```
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

9.34.3.5 hpss_HashCreate()

```
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.

9.34.3.6 hpss_HashDecode()

```
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	<i>Buffer</i>	Buffer used for the decode
in	<i>Length</i>	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.

9.34.3.7 hpss_HashDelete()

```
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	<i>Hash</i>	Hash Engine

Return values

0	Success
<i>HPSS_EFAULT</i>	Invalid hash context or buffer

Return values

<i>HPSS_EINVAL</i>	Invalid hash type

9.34.3.8 hpss_HashDigestLength()

```
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

<i>HPSS_EINVAL</i>	Invalid hash type

9.34.3.9 hpss_HashDigestToHex()

```
void hpss_HashDigestToHex (
    unsigned char * digest,
    int digest_len,
    unsigned char * digest_str )
```

Convert a hash digest to hex.

Parameters

in	<i>digest</i>	Digest String (binary representation)
in	<i>digest_len</i>	Digest String length
in, out	<i>digest_str</i>	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.

9.34.3.10 hpss_HashDigestToString()

```
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	<i>Buffer</i>	Buffer used for the digest binary data
in	<i>Length</i>	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

9.34.3.11 hpss_HashDuplicate()

```
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	<i>Hash</i>	Hash Engine

Returns

An identical copy of the hash, or NULL if the hash cannot be duplicated.

9.34.3.12 hpss_HashEncode()

```
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	<i>Hash</i>	Hash reference
in, out	<i>Buffer</i>	Buffer used for the encoded hash
in	<i>Type</i>	Algorithm Type

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EFAULT</i>	Invalid hash context or buffer pointer
<i>HPSS_EINVAL</i>	Invalid hash provided

9.34.3.13 hpss_HashExtract()

```
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	<i>Hash</i>	Hash Engine
in, out	<i>Buffer</i>	Buffer containing the hash engine
in, out	<i>Length</i>	Buffer length
in, out	<i>Type</i>	Hash Engine Type

Return values

0	Success
<i>HPSS_EFAULT</i>	Invalid hash context, buffer or length pointer
<i>HPSS_EINVAL</i>	Invalid hash provided

9.34.3.14 hpss_HashFinish()

```
unsigned char * hpss_HashFinish (
    hpss_hash_t Hash,
    unsigned int * Length )
```

Finish hashing and return the result.

Parameters

in	<i>Hash</i>	Hash Engine
in, out	<i>Length</i>	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.

9.34.3.15 hpss_HashFinishDigest()

```
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	<i>Hash</i>	Pointer to hash reference
out	<i>Digest</i>	Pointer to allocated memory for binary digest
in	<i>Length</i>	Size of allocated memory for digest

Returns

The size of the digest, or a negative value on failure

Return values

<i>HPSS_EINVAL</i>	Invalid hash type
<i>HPSS_ENOMEM</i>	Memory allocation failure
<i>HPSS_ESYSTEM</i>	Hash engine system failure

Note

The digest returned is in binary format.

9.34.3.16 hpss_HashFinishHex()

```
char * hpss_HashFinishHex (
    hpss_hash_t Hash )
```

Return a finished hash as a hex string.

Parameters

in	<i>Hash</i>	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.

9.34.3.17 hpss_HashGetHWFlags()

```
int hpss_HashGetHWFlags ( )
```

Return checksum flags.

Returns information about certain hardware-acceleration available for checksumming on this system.

Return values

<i>HPSS_CKSUM_AFALG</i>	Kernel-level checksumming available
<i>HPSS_CKSUM_SSE42</i>	x86 SSE42 hardware acceleration available
<i>HPSS_CKSUM_VPMSUM</i>	ppc hardware acceleration available

9.34.3.18 hpss_HashLoad()

```
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	<i>Buffer</i>	Data Buffer
in	<i>BufferLen</i>	Buffer Length
in	<i>type</i>	Hash Type

Returns

An HPSS hash loaded with the buffer contents, or NULL if the buffer cannot be loaded.

9.34.3.19 hpss_HashReset()

```
int hpss_HashReset (
    hpss_hash_t Hash )
```

Reset hash engine in preparation for a new stream.

Parameters

in	<i>Hash</i>	Hash Engine

Return values

0	Success
<i>HPSS_EFAULT</i>	Invalid hash context
<i>HPSS_ESYSTEM</i>	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.

9.34.3.20 hpss_HashType()

```
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	<i>Hash</i>	Hash reference
out	<i>Type</i>	Pointer to returned algorithm type

Return values

0	Success
HPSS_EINVAL	Invalid hash type
HPSS_EFAULT	Invalid hash or type pointer

9.34.3.21 hpss_HexStringToHashDigest()

```
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	<i>HexString</i>	String of hex digits
out	<i>Length</i>	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

9.35 HPSS Config File API

Configuration File Parser APIs.

Classes

- struct [hpss_cfg_stanza](#)
Configuration stanza.

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 key↵ OrValue, 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 * hpsscfx_ConfParse (char cfgFile[])`
Parse the configuration file.

9.35.1 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 subanzas (`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.

9.35.2 Class Documentation

9.35.2.1 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)

9.35.3 Macro Definition Documentation

9.35.3.1 HPSS_CFG_MAX_CHARS_PER_LINE

```
#define HPSS_CFG_MAX_CHARS_PER_LINE 512 /* HPSS.conf spec is 128 */
```

Maximum characters per line

9.35.3.2 HPSS_CFG_MAX_LEVEL

```
#define HPSS_CFG_MAX_LEVEL 10
```

Maximum number of levels supported

9.35.4 Typedef Documentation

9.35.4.1 hpss_cfg_stanza_t

```
typedef struct hpss\_cfg\_stanza hpss_cfg_stanza_t
```

Configuration stanza.

Stanza structure

9.35.5 Enumeration Type Documentation

9.35.5.1 HPSS_CFG_STANZA_TYPE

```
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

9.35.6 Function Documentation

9.35.6.1 hpss_CfgFindKey()

```
hpss_cfg_stanza_t * hpss_CfgFindKey (
    char const * searchStr,
    hpss_cfg_stanza_t * cfgStart,
    int caseFlag )
```

Search for a Key value.

Parameters

in	<i>searchStr</i>	Key to search for
in	<i>cfgStart</i>	Head of level to search
in	<i>caseFlag</i>	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.

9.35.6.2 hpss_CfgFindToken()

```
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	<i>searchToken</i>	Token to search for
in	<i>cfgStart</i>	Head of level to search
in	<i>keyOrValue</i>	Search type: SEARCH_BY_KEY or SEARCH_BY_VALUE
in	<i>subLevels</i>	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_↵
LEVEL search this level only SEARCH_SUB_LEVELS search sublevels too

This function calls itself recursively if [subLevels] is non-zero

9.35.6.3 hpss_CfgFree()

```
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.

9.35.6.4 hpss_CfgParse()

```
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.

9.35.6.5 hpsscfgx_ConfParse()

```
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.

9.36 HPSS Access Control List APIs

APIs for manipulating ACLs using the HACL Library.

Classes

- struct [hacl_acl_entry](#)
Defines an access control list entry.

Macros

- #define [HACL_M_ALLOW_UNAUTHENTICATED](#) (0x10000000)
Allow unauthenticated 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.

9.36.1 Detailed Description

APIs for manipulating ACLs using the HACL Library.

9.36.2 Class Documentation

9.36.2.1 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[128]	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[256]	Realm name which qualifies user and group names.

9.36.3 Macro Definition Documentation

9.36.3.1 HACL_MAX_ENTRY_SIZE

```
#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.

9.36.3.2 HPSS_EHACL_UNAUTH

```
#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).

9.36.4 Typedef Documentation

9.36.4.1 hacl_acl_entry_t

```
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.

9.36.5 Function Documentation

9.36.5.1 hacl_ConvertACLToHACL()

```
int hacl_ConvertACLToHACL (
    ns_ACLConfArray_t * ACL_n,
    hacl_acl_t ** ACL_h )
```

Convert ACL to string format.

Parameters

in	<i>ACL</i> _{<i>n</i>}	ACL to convert
out	<i>ACL</i> _{<i>h</i>}	Converted ACL

Function `hacL_ConvertACLToHACL` converts ACLs from nameserver format into string format.

Return values

<i>0</i>	Success; ACL_h contains converted ACL
<i>EINVAL</i>	Invalid arguments (e.g., unknown ACL entry type)
<i>ENOMEM</i>	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.

9.36.5.2 hacL_ConvertHACLPermsToPerms()

```
int hacL_ConvertHACLPermsToPerms (
    char * HPerms,
    unsigned char * Perms )
```

Convert permission string to HPSS format.

Parameters

in	<i>HPerms</i>	Permissions to Convert
out	<i>Perms</i>	Converted Permissions

Function `hacL_ConvertHACLPermsToPerms` converts a permission string to HPSS nameserver format. The string can contain any of the characters "rwxidRWXCID-" in any order. Hyphens are ignored. Other characters are used to control which bit positions get set in Perms.

Return values

<i>0</i>	Success
<i>EINVAL</i>	HPerms contains one or more invalid characters

Note

HPerms must be a null-terminated string.

9.36.5.3 hacl_ConvertHACLToACL()

```
int hacl_ConvertHACLToACL (
    hacl_acl_t * ACL_h,
    ns_ACLConfArray_t * ACL_n )
```

Convert ACL to nameserver format.

Parameters

in	<i>ACL</i> ↔ <i>_h</i>	ACL to Convert
out	<i>ACL</i> ↔ <i>_n</i>	Converted ACL

Return values

0	Success; ACL_n contains converted ACL
<i>EINVAL</i>	Invalid arguments
<i>ENOMEM</i>	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).

9.36.5.4 hacl_ConvertHACLToString()

```
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	<i>ACL_h</i>	ACL to Convert
in	<i>Flags</i>	Flags
in	<i>EntrySeparator</i>	Entry Separator
out	<i>ACL_s</i>	Converted ACL

Return values

0	Success
<i>EINVAL</i>	Invalid arguments

Return values

<i>ENOMEM</i>	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_REALM` 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.

9.36.5.5 `hacl_ConvertHACLTypeToType()`

```
int hacl_ConvertHACLTypeToType (
    char * HType,
    unsigned char * Type )
```

Convert ACL entry type to HPSS format.

Parameters

in	<i>HType</i>	Type to Convert
out	<i>Type</i>	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
<i>EINVAL</i>	HType was not a recognized ACL type

9.36.5.6 hacl_ConvertPermsToHACLPerms()

```
int hacl_ConvertPermsToHACLPerms (
    unsigned char Perms,
    char * HPerms )
```

Convert HPSS permission mask to string.

Parameters

in	<i>Perms</i>	Permissions to Convert
out	<i>HPerms</i>	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
<i>EINVAL</i>	HPerms is NULL

Note

Caller supplies storage for HPerms sufficient to hold at least HACL_MAX_PERMS characters.

9.36.5.7 hacl_ConvertStringsToHACL()

```
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	<i>NumEntries</i>	Number of array elements
in	<i>Entries</i>	Array of ACL strings
in	<i>DefaultRealm</i>	Default Realm
in	<i>WhiteSpaceChars</i>	ACL entry separators
in	<i>Flags</i>	Flags
out	<i>ACL_h</i>	Access Control List

Function hacl_ConvertStringsToHACL converts one or more strings representing acl entries into a single access control list.

Return values

<i>0</i>	The function succeeded
<i>EINVAL</i>	An ACL entry was not in the right format
<i>ENAMETOOLONG</i>	Default realm name is too long
<i>ENOMEM</i>	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.)

9.36.5.8 hacl_ConvertTypeToHACLType()

```
int hacl_ConvertTypeToHACLType (
    unsigned char Type,
    char * HType )
```

Convert ACL entry type to string.

Parameters

in	<i>Type</i>	ACL entry type to convert
out	<i>HType</i>	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

<i>0</i>	Success
<i>EINVAL</i>	Type argument is invalid

Note

Caller supplies storage for HType sufficient to hold at least HACL_MAX_TYPE characters.

9.36.5.9 hacl_DeleteHACL()

```
int hacl_DeleteHACL (
    char * Path,
    unsigned32 Options,
    hacl_acl_t * ACL_h )
```

Deletes selected ACL entries from the named object.

Parameters

in	<i>Path</i>	Path of Object
in	<i>Options</i>	Processing Options
in	<i>ACL</i> ↔ <i>_h</i>	ACL to be removed

Return values

0	Success
<i>Other</i>	Error code indicating the nature of the error

9.36.5.10 hacl_GetHACL()

```
int hacl_GetHACL (
    char * Path,
    unsigned32 Options,
    hacl_acl_t ** ACL_h )
```

Query HPSS to get the ACL for the named object.

Parameters

in	<i>Path</i>	Path of Object
in	<i>Options</i>	Processing Options
in	<i>ACL</i> ↔ <i>_h</i>	Object ACL

Return values

0	Success
<i>EINVAL</i>	Invalid arguments

9.36.5.11 hacl_SetHACL()

```
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	<i>Path</i>	Path of Object
in	<i>Options</i>	Processing Options
in	<i>ACL</i> ↔ <i>_h</i>	New ACL

Return values

0	Success
<i>Other</i>	Error indicating the nature of the error

9.36.5.12 hacl_SortHACL()

```

void hacl_SortHACL (
    hacl_acl_t * ACL_h )

```

Sort an ACL into canonical order.

Parameters

in, out	<i>ACL</i> ↔ <i>_h</i>	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.

9.36.5.13 hacl_UpdateHACL()

```

int hacl_UpdateHACL (
    char * Path,
    unsigned32 Options,
    hacl_acl_t * ACL_h )

```

Change the selected ACL entries for the named object.

Parameters

in	<i>Path</i>	Path of Object
in	<i>Options</i>	Processing Options
in	<i>ACL</i> _↔ <i>_h</i>	ACL to be updated

Return values

0	Success
<i>Other</i>	Error indicating the nature of the failure

9.37 Gatekeeper Site Library

Prototypes for HPSS Gatekeeper site interfaces.

Classes

- struct [gk_EntryInfo](#)
Gatekeeper Request Information Entry.
- struct [gk_ReqInfo](#)
Gatekeeper RTM info.

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_uid_t ControlNo)
Called when an HPSS file is being opened by an Authorized Caller.
- signed32 [gk_site_CreateComplete](#) (hpss_uid_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.

9.37.1 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.

9.37.2 Class Documentation

9.37.2.1 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_RequestType_t	RequestType	Request Type.
gk_ReqInfo_t *	RTMReqInfoP	RTM Request Information. Real-time monitoring request information for retry logic
gk_Stage_t	StageSpecificInfo	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.

9.37.2.2 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

9.37.3 Typedef Documentation

9.37.3.1 gk_EntryInfo_t

```
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.

9.37.3.2 gk_ReqInfo_t

```
typedef struct gk_ReqInfo gk_ReqInfo_t
```

Gatekeeper RTM info.

The following structure contains information about an RTM request entry.

9.37.4 Function Documentation

9.37.4.1 gk_site_Close()

```
signed32 gk_site_Close (
    hpss_uuid_t ControlNo )
```

Called when an HPSS file is being opened by an Authorized Caller.

Parameters

in	<i>ControlNo</i>	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](#)

9.37.4.2 gk_site_CreateComplete()

```
signed32 gk_site_CreateComplete (
    hpss_uuid_t ControlNo )
```

Called when an HPSS file create has been completed.

Parameters

in	<i>ControlNo</i>	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).

```

*

9.37.4.3 gk_site_CreateStats()

```

void gk_site_CreateStats (
    gk_EntryInfo_t EntryInfo )

```

Called when an HPSS file is being created by an Authorized Caller.

Parameters

in	<i>EntryInfo</i>	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_CREATE`.

```

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.

```

*

9.37.4.4 gk_site_GetMonitorTypes()

```

signed32 gk_site_GetMonitorTypes (
    u_signed64 * MonitorTypeBitsP )

```

Called to get the request types being monitored.

Parameters

in	<i>MonitorTypeBitsP</i>	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*.

9.37.4.5 gk_site_Init()

```
signed32 gk_site_Init (
    char * SitePolicyPathNameP )
```

Called to initialize the Site Interface.

Parameters

in	<i>SitePolicyPathNameP</i>	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 *SitePolicyPathNameP* - 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_MonitorTypeBits* = [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 Log File specified in *SitePolicyPathNameP* (optional) Initialize mutexes, structures, etc. (eg May want to initialize cache(s) associated with requests.)

9.37.4.6 gk_site_Open()

```
signed32 gk_site_Open (
    gk_EntryInfo_t EntryInfo,
    unsigned32 * WaitTimeP )
```

Called when an HPSS file is being opened.

Parameters

in	<i>EntryInfo</i>	Information about the open
out	<i>WaitTimeP</i>	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

9.37.4.7 gk_site_OpenStats()

```
void gk_site_OpenStats (
    gk_EntryInfo_t EntryInfo )
```

Called when an HPSS file is being opened by an Authorized Caller.

Parameters

in	<i>EntryInfo</i>	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_OPEN.

See also

[gk_site_CreateStats](#)

9.37.4.8 gk_site_Stage()

```
signed32 gk_site_Stage (
    gk_EntryInfo_t EntryInfo,
    unsigned32 * WaitTimeP )
```

Called when an HPSS file is being staged.

Parameters

in	<i>EntryInfo</i>	Information about the stage
out	<i>WaitTimeP</i>	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](#)

9.37.4.9 gk_site_StageComplete()

```
signed32 gk_site_StageComplete (
    hpss_uuid_t ControlNo )
```

Called when an HPSS file stage has completed.

Parameters

<i>in</i>	<i>ControlNo</i>	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

<i>0</i>	Success

Note

This routine is only called when monitoring `GK_MONITOR_STAGE`

See also

[gk_site_CreateComplete](#)

9.37.4.10 gk_site_StageStats()

```
void gk_site_StageStats (
    gk_EntryInfo_t EntryInfo )
```

Called when an HPSS file is being staged by an Authorized Caller. Authorized Caller.

Parameters

<i>in</i>	<i>EntryInfo</i>	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_STAGE`.

See also

[gk_site_CreateStats](#)

9.38 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 *OutAcct, unsigned32 *OkToCache)
Validate a user's account index.
- signed32 [av_site_ValidateChacct](#) (hpss_connect_handle_t Binding, hpss_reqid_t RequestId, unsigned32 UserRealmId, unsigned32 UserId, 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 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.
- 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.

9.38.1 Detailed Description

Account Validation Site Library Prototypes.

9.38.2 Function Documentation

9.38.2.1 av_site_AcctIdxToName()

```
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	<i>Binding</i>	RPC Binding Info
in	<i>RequestId</i>	Id of this request
in	<i>RealmId</i>	User's Realm Id
in	<i>Uid</i>	User id
in	<i>Gid</i>	User's Group Id
in	<i>Flags</i>	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	<i>Acct</i>	Account Index to Convert
out	<i>AcctName</i>	Name of the Account
out	<i>OkToCache</i>	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

<i>HPSS_EUSEDEFAULT</i>	If the routine is not implemented by the site, use the corresponding default routine
<i>0</i>	Success
<i>HPSS_EBUSY</i>	The server is busy. The request should be tried again later.
<i>HPSS_ENOENT</i>	That account index does not exist
<i>HPSS_EPERM</i>	User is not a member of that account
<i>HPSS_ESYSTEM</i>	This routine received an internal error; usually a site customization problem.
<i><0</i>	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.

9.38.2.2 av_site_AcctNameToIdx()

```

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	<i>Binding</i>	RPC Binding Information for this Request
in	<i>RequestId</i>	Id of this Request
in	<i>RealmId</i>	Realm Id of the User
in	<i>Uid</i>	User's Id (unix uid)
in	<i>Gid</i>	Group Id of the user/file (unix gid)
in	<i>Flags</i>	Zero or more of the following bit flags: ACCT_FLAGS_VALID_INDEX - Make sure this user is allowed to use this account.
in	<i>InAcctName</i>	Optional Account Name to convert
out	<i>OutAcctName</i>	Returned name of the Account
out	<i>Acct</i>	Returned account index for this account
out	<i>OkToCache</i>	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. Im-

plementation 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

<code>HPSS_EUSEDEFAULT</code>	If this routine is not implemented by the site, use the corresponding default routine.
<code>0</code>	Success
<code>HPSS_EBUSY</code>	The server is busy. The request should be retried later.
<code>HPSS_ENOENT</code>	That account index does not exist
<code>HPSS_EPERM</code>	User is not a member of that account.
<code>HPSS_ESYSTEM</code>	This routine received an internal error. Usually a site customization problem.
<code><0</code>	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;
}

```

9.38.2.3 av_site_Initialize()

```

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	<i>AcctPolicy</i>	Account Policy metadata record

Return values

0	Success
<i>HPSS_ESYSTEM</i>	Site code cannot initialize properly
<i>HPSS_ENOMEM</i>	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;  
}
```

9.38.2.4 av_site_Shutdown()

```
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 guaranteed to be called).

Return values

0	Success
<i>HPSS_EINVAL</i>	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;  
}
```

9.38.2.5 av_site_ValidateAccount()

```
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	<i>Binding</i>	RPC binding information
in	<i>RequestId</i>	Id of this Request
in	<i>RealmId</i>	User's Realm Id
in	<i>Uid</i>	User's Id (unix uid)
in	<i>Acct</i>	Account Index to validate
in	<i>Flags</i>	Zero or more of the following bit flags: • ACCT_VALIDATE_FLAGS_CREATING - A file or directory being created.
in	<i>ParentAcct</i>	Account index of parent directory
out	<i>OutAcct</i>	Account index to use
out	<i>OkToCache</i>	TRUE if this information may be cached by the client.

Return values

<i>HPSS_EUSEDEFAULT</i>	If the routine is not implemented by the site, use the corresponding default routine
<i>0</i>	Success
<i>HPSS_EBUSY</i>	The server is busy. The request should be tried again later.
<i>HPSS_ENOENT</i>	That account index does not exist
<i>HPSS_EPERM</i>	User is not a member of that account
<i>HPSS_ESYSTEM</i>	This routine received an internal error; usually a site customization problem.
<i><0</i>	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 information 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;
}

```

9.38.2.6 av_site_ValidateChacct()

```

signed32 av_site_ValidateChacct (
    hpss_connect_handle_t Binding,
    hpss_reqid_t RequestId,
    unsigned32 UserRealmId,
    unsigned32 UserId,
    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	<i>Binding</i>	RPC binding information
in	<i>RequestId</i>	Id of this Request
in	<i>User↔ RealmId</i>	Realm Id of the User Performing This Op
in	<i>UserId</i>	User Id (unix uid) of the User Performing This Op
in	<i>FileRealmId</i>	Realm Id of the file owner
in	<i>FileUid</i>	User Id (unix uid) of the file owner
in	<i>FileGid</i>	Group Id (unix gid) of the file
in	<i>OldAcct</i>	Old account index for the file
in	<i>NewAcct</i>	New account index for the file
out	<i>OutAcct</i>	Account index for the file
out	<i>OkToCache</i>	TRUE if this information may be cached by the client.

Return values

<i>HPSS_EUSEDEFAULT</i>	If the routine is not implemented by the site, use the corresponding default routine
<i>0</i>	Success
<i>HPSS_EBUSY</i>	The server is busy. The request should be tried again later.
<i>HPSS_ENOENT</i>	That account index does not exist
<i>HPSS_EPERM</i>	User is not a member of that account
<i>HPSS_ESYSTEM</i>	This routine received an internal error; usually a site customization problem.
<i><0</i>	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;
}

```

9.38.2.7 `av_site_ValidateChown()`

```

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	<i>Binding</i>	RPC binding information
in	<i>RequestId</i>	Id of this Request
in	<i>OldRealmId</i>	Old Realm Id of the file owner
in	<i>OldUid</i>	Old User Id (unix uid) of the file owner
in	<i>OldGid</i>	Old Group Id (unix gid) of the file
in	<i>OldAcct</i>	Old account index of the file
in	<i>New↔ RealmId</i>	New Realm Id of the file owner
in	<i>NewUid</i>	New User Id (unix uid) of the file owner
in	<i>NewGid</i>	New Group Id (unix gid) of the file
in	<i>SessionAcct</i>	Current session account index of user
out	<i>OutAcct</i>	Account index to use for the file
out	<i>OkToCache</i>	TRUE if this information may be cached by the client.

Return values

<i>HPSS_EUSEDEFAULT</i>	If the routine is not implemented by the site, use the corresponding default routine
<i>0</i>	Success
<i>HPSS_EBUSY</i>	The server is busy. The request should be tried again later.
<i>HPSS_ENOENT</i>	That account index does not exist
<i>HPSS_EPERM</i>	User is not a member of that account
<i>HPSS_ESYSTEM</i>	This routine received an internal error; usually a site customization problem.
<i><0</i>	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 information 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;
}
```

9.38.2.8 av_site_ValidateCreate()

```
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	<i>Binding</i>	RPC binding information
in	<i>RequestId</i>	Id of this Request
in	<i>RealmId</i>	New file owner's Realm Id
in	<i>Uid</i>	User Id (unix uid) of the new file's owner
in	<i>Gid</i>	Group Id (unix gid) of the new file
in	<i>Acct</i>	Account index of the new file
in	<i>ParentAcct</i>	Account index of the new file's parent directory
in	<i>ParentAttrs</i>	Parent Directory Attributes including: • <code>CORE_ATTR_MODE_PERMS</code>
out	<i>OutAcct</i>	Account index to use for the file
out	<i>OkToCache</i>	TRUE if this information may be cached by the client.

Return values

<i>HPSS_EUSEDEFAULT</i>	If this routine is not implemented by the site, use the corresponding default routine.
<i>0</i>	Success
<i>HPSS_EBUSY</i>	The server is busy. The request should be tried again later.
<i>HPSS_ENOENT</i>	That account index does not exist
<i>HPSS_EPERM</i>	User is not a member of that account
<i>HPSS_ESYSTEM</i>	This routine received an internal error; usually a site customization problem.
<i><0</i>	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

File Documentation

10.1 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 *OutAcct, unsigned32 *OkToCache)
Validate a user's account index.
- signed32 [av_site_ValidateChacct](#) (hpss_connect_handle_t Binding, hpss_reqid_t RequestId, unsigned32 UserRealmId, unsigned32 UserId, 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 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.
- 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.

10.2 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 *OutAcct, unsigned32 *OkToCache)
Validate a user's account index.
- signed32 [av_site_ValidateChacct](#) (hpss_connect_handle_t Binding, hpss_reqid_t RequestId, unsigned32 UserRealmId, unsigned32 UserId, 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 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.
- 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.

10.3 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.

10.3.1 Detailed Description

Functions for determining file access.

10.4 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 CoreID, 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.

10.4.1 Detailed Description

Functions for manipulating and translating account information.

10.4.2 Function Documentation

10.4.2.1 API_DetermineAcct()

```
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	<i>UserCred</i>	user credentials
in	<i>ThreadContext</i>	thread context
in	<i>CoreServerID</i>	id of Core Server
in	<i>RequestID</i>	request id
in	<i>SiteId</i>	sites location
out	<i>AcctCode</i>	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).

10.4.2.2 API_GetAcctForParent()

```
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	<i>ThreadContext</i>	thread context
in	<i>RequestID</i>	request id
in	<i>ParentHandle</i>	directory handle
out	<i>AcctCode</i>	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()

10.4.2.3 API_PutAcctForParent()

```
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 specified account code for the specified parent handle.

Parameters

in	<i>ParentHandle</i>	directory handle
in	<i>AcctCode</i>	account code

Return values

<i>-EINVAL</i>	Not a directory handle
<i>-ENOMEM</i>	Unable to allocate memory.
<i>0</i>	No error. Account code is cached

10.4.2.4 API_SetCurrentSessionAcct()

```
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	<i>ThreadContext</i>	thread context
in	<i>SiteId</i>	site id

Return values

<i>-EFAULT</i>	SiteId or threadcontext pointer is not valid.
<i>0</i>	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.

10.5 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.

10.5.1 Detailed Description

Functions for manipulating access control lists.

10.6 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.

10.6.1 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.

10.6.2 Function Documentation

10.6.2.1 API_AddFSDirCache()

```
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	<i>User</i>	Ucred username
in	<i>FSID</i>	Deleted Object's FSID
in	<i>TrashHandle</i>	Trashcan Directory Handle
in	<i>TrashContainer</i>	Trashcan Container (note for fs this is the fsroot)

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINVAL</i>	Cache is not initialized or invalid fsid
<i>HPSS_ENOMEM</i>	Out of memory
<i>HPSS_ENOSPACE</i>	Cache is full; couldn't purge down

10.6.2.2 API_AddHomeDirCache()

```
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	<i>User</i>	Ucred username
in	<i>FSID</i>	Homedir Fileset Id
in	<i>TrashHandle</i>	Trashcan Directory Handle
in	<i>TrashContainer</i>	Trashcan Container (note for homedirs this is the homedir)

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINVAL</i>	Cache is not initialized or invalid fsid
<i>HPSS_ENOMEM</i>	Out of memory
<i>HPSS_ENOSPACE</i>	Cache is full; couldn't purge down

10.6.2.3 API_ClearFSDirCache()

```
int32_t API_ClearFSDirCache ( )
```

Remove all entries from the cache.

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINAL</i>	Cache is not initialized

Note

This function completely frees all cache resources, and then reinitializes an empty tree for future processing.

10.6.2.4 API_ClearHomeDirCache()

```
int32_t API_ClearHomeDirCache ( )
```

Remove all entries from the cache.

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINVAL</i>	Cache is not initialized

Note

This function completely frees all cache resources, and then reinitializes an empty tree for future processing.

10.6.2.5 API_GetFSDirCache()

```
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	<i>User</i>	User ucred; This a partial lookup key
in	<i>FSID</i>	Parent FSID; partial lookup key
in, out	<i>TrashHandle</i>	Trashcan Handle
in, out	<i>TrashContainer</i>	Trashcan Container

Return values

<i>HPSS_EINVAL</i>	Cache is not initialized
<i>HPSS_ENOENT</i>	No entry found
<i>HPSS_E_NOERROR</i>	Success

10.6.2.6 API_GetHomeDirCache()

```
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

<i>in</i>	<i>User</i>	User ucred; This is the lookup key
<i>in, out</i>	<i>FSID</i>	Fileset Id
<i>in, out</i>	<i>TrashHandle</i>	Trashcan Handle
<i>in, out</i>	<i>TrashContainer</i>	Trashcan Container

Return values

<i>HPSS_EINVAL</i>	Cache is not initialized
<i>HPSS_ENOENT</i>	No entry found
<i>HPSS_E_NOERROR</i>	Success

10.6.2.7 API_GetJunctionCache()

```
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.

10.6.2.8 API_InitFSDirCache()

```
int32_t API_InitFSDirCache ( )
```

Initialize the fs trashcan cache.

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_ENOMEM</i>	Out of memory

10.6.2.9 API_InitHomeDirCache()

```
int32_t API_InitHomeDirCache ( )
```

Initialize the home directory trashcan cache.

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_ENOMEM</i>	Out of memory

10.6.2.10 API_RebuildJunctionCache()

```
int32_t API_RebuildJunctionCache ( )
```

Recreate the junction cache.

This function releases the junction cache and then rebuilds it in full.

10.6.2.11 API_RemoveFSDirCache()

```
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	<i>User</i>	Ucred username
in	<i>FSID</i>	Deleted object's fileset id

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINVAL</i>	Cache is not initialized
<i>HPSS_ENOMEM</i>	Out of memory

10.6.2.12 API_RemoveHomeDirCache()

```
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	<i>User</i>	Ucred username

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINVAL</i>	Cache is not initialized
<i>HPSS_ENOMEM</i>	Out of memory

10.6.2.13 fsrootkeycompare()

```
int fsrootkeycompare (
    const hpss_irbtree_node_t * t1,
    const hpss_irbtree_node_t * t2 )
```

Compare two cached fs handles.

Parameters

in	<i>t1</i>	irbTree node that intrudes upon first fs cache
in	<i>t2</i>	irbTree node that intrudes upon second fs cache

Return values

<i>0</i>	cache entries are equal
<i>-1</i>	cache entry 1 comes before 2
<i>1</i>	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.

10.6.2.14 stale_thresh_cmp_factor()

```
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	<i>t1</i>	irbTree node that corresponds to trash cache entry
in	<i>t2</i>	irbTree node object provided by the arbitrary comparison function (unused)
in	<i>factor</i>	cutoff factor, increase to lower the cutoff time

Return values

<i>FALSE</i>	stale
<i>TRUE</i>	not stale

10.6.2.15 thresh_cmp()

```
int32_t thresh_cmp (
    trash_cache_t * entry,
    time_t cutoff )
```

Determines whether an trash cache entry is stale.

Parameters

in	<i>entry</i>	trash cache entry
in	<i>cutoff</i>	Time past an entry would be stale

Return values

<i>FALSE</i>	stale
<i>TRUE</i>	not stale

10.6.2.16 userkeycompare()

```
int32_t userkeycompare (
    const hpss_irbtree_node_t * d1,
    const hpss_irbtree_node_t * d2 )
```

Compare two cached homedir handles.

Parameters

in	<i>d1</i>	irbTree node that intrudes upon first cache entry
in	<i>d2</i>	irbTree node that intrudes upon second cache entry

Return values

0	names are equal
-1	h1 comes before h2
1	h1 comes after h2

10.7 `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.

10.7.1 Detailed Description

Functions for changing directories.

10.8 `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.

10.8.1 Detailed Description

Functions for modifying object mode bits.

10.9 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.

10.9.1 Detailed Description

Functions for changing the owner and/or group of an object.

10.10 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.

10.10.1 Detailed Description

Functions for modifying the default root location for a client.

10.10.2 Function Documentation

10.10.2.1 API_ApplyRootDir()

```
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	<i>Path</i>	path to the object
in	<i>RootDir</i>	root dir path
out	<i>NewPath</i>	path rel root dir
in	<i>ThreadContext</i>	current context

Return values

<i>-ENAMETOOLONG</i>	The provided pathname is too long, or the constructed pathname is too long.

10.11 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.

10.11.1 Detailed Description

Functions for closing open files.

10.12 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.

10.12.1 Detailed Description

Functions for closing open directories.

10.13 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.

10.13.1 Detailed Description

Functions for copying one file to another.

10.14 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.

10.14.1 Detailed Description

Functions for setting the COS of a file.

10.15 [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.

10.15.1 Detailed Description

Functions for retrieving or inserting file table information.

10.16 [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](#) *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_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](#) *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_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.

10.16.1 Detailed Description

Functions for retrieving file attributes.

10.16.2 Function Documentation

10.16.2.1 API_ChecksumStateToFlags()

```
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	<i>Value</i>	Checksum state string

Return values

≥ 0	Digest flags structure based upon the input state string
< 0	Error indicating the nature of the error

10.16.2.2 API_GetMultiFileDigest()

```
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	<i>Fildes</i>	Open bitfile descriptor
out	<i>StripeLength</i>	Digest stripe length
out	<i>DigestList</i>	Returned hash digest information

Return values

0	Success

Return values

<code>-EBADF</code>	Fildes is not a valid open descriptor
<code>-ENOENT</code>	Digest information doesn't exist
<code>-EBUSY</code>	Fildes is a busy descriptor
<code>-ESTALE</code>	Fildes is a stale descriptor and should be reopened
<code>-EFAULT</code>	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 converted 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](#)

10.16.2.3 API_GetMultiFileDigestHandle()

```
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	<i>ThreadContext</i>	API Thread Context
in	<i>ObjHandle</i>	Object Handle
in	<i>Path</i>	File Path
in	<i>Ucred</i>	User Credentials
out	<i>StripeLength</i>	Digest stripe length
out	<i>DigestList</i>	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 converted 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](#)

10.16.2.4 API_GetSingleFileDigestHandle()

```
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	<i>ThreadContext</i>	API thread context
in	<i>*ObjHandle</i>	parent object handle
in	<i>*Path</i>	path of file to get attributes
in	<i>*Ucred</i>	user credentials
out	<i>*DigestList</i>	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 indicate 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](#)

10.16.2.5 API_UDAFindKey()

```
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	<i>Attrs</i>	User-defined Attributes
in	<i>Key</i>	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.

10.17 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_svr_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 *FilesetHandle)
Deletes an existing fileset.
- int [hpss_FilesetGetAttributes](#) (const char *FilesetName, const uint64_t *FilesetId, const ns_ObjHandle_t *FilesetHandle, const hpss_svr_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.

10.17.1 Detailed Description

Functions for manipulating filesets.

10.18 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 SelfFlags, 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 SelfFlags, 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 SelfFlags, 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 SelfFlagsIn, const hpss_vattr_t *AttrIn, uint64_t *SelfFlagsOut, 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.

10.18.1 Detailed Description

Functions for setting file attributes.

10.19 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.

10.19.1 Detailed Description

Functions for retrieving the current working directory.

10.20 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.

10.20.1 Detailed Description

Functions for finding HPSS trashcans.

10.21 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_SitIdToName](#) (const hpss_id_t *SitId, char *SiteName)
Convert Site ID to Site Name.
- int [hpss_SiteNameToId](#) (const char *SiteName, hpss_id_t *SitId)
Convert Site Name to Site ID.

10.21.1 Detailed Description

Functions for mapping between names and ids.

10.22 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_cstub.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

- [int32_t _internal_hpss_LocateRootCS](#) (hpss_reqid_t RequestId, ls_map_t *Location)
Internal function for locating the root core server.
- [int32_t _internal_hpss_LocateServerByID](#) (hpss_reqid_t RequestId, hpss_srvr_id_t ServerId, ls_map_t *Location)
Internal function for locating servers by ID.
- [int32_t _internal_hpss_LocateServersByType](#) (hpss_reqid_t RequestId, hpss_id_t SiteId, uint32_t ServerType, uint32_t SubsystemId, ls_map_array_t **Locations)
Internal function for locating servers by type.
- [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, apicsepio_t *CSEPIOInfo, 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, apicsepio_t *CSEPIOInfo, 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

10.22.1 Detailed Description

Functions for initializing the Client API.

10.22.2 Function Documentation

10.22.2.1 [_internal_hpss_LocateRootCS\(\)](#)

```
int32_t _internal_hpss_LocateRootCS (
    hpss_reqid_t RequestId,
    ls_map_t * Location )
```

Internal function for locating the root core server.

Note

This function is the same as [hpss_LocateRootCS](#), but it does not include initialization of the API. This can be used in situations where the API has not been initialized yet. See [hpss_LocateRootCS](#) for details.

10.22.2.2 `_internal_hpss_LocateServerByID()`

```
int32_t _internal_hpss_LocateServerByID (
    hpss_reqid_t RequestId,
    hpss_srvr_id_t ServerId,
    ls_map_t * Location )
```

Internal function for locating servers by ID.

Note

This function is the same as `hpss_LocateServerByID`, but it does not include initialization of the API. This can be used in situations where the API has not been initialized yet. See `hpss_LocateServerByID` for details.

10.22.2.3 `_internal_hpss_LocateServersByType()`

```
int32_t _internal_hpss_LocateServersByType (
    hpss_reqid_t RequestId,
    hpss_id_t SiteId,
    uint32_t ServerType,
    uint32_t SubsystemId,
    ls_map_array_t ** Locations )
```

Internal function for locating servers by type.

Note

This function is the same as `hpss_LocateServerByType`, but it does not include initialization of the API. This can be used in situations where the API has not been initialized yet. See `hpss_LocateServerByType` for details.

10.22.2.4 `API_ClientAPIInit()`

```
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	<i>ThreadContext</i>	state of thread

Return values

0	No error. Thread specific state is valid.

10.22.2.5 API_DebugPrintError()

```
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	<i>File</i>	Pointer to a FILE structure
in	<i>RequestID</i>	Pointer to a request id
in	<i>Format</i>	Format-control string

Return values

None.	

10.22.2.6 API_DebugPrintRequest()

```
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	<i>File</i>	Pointer to a FILE structure
in	<i>RequestID</i>	Pointer to a request id
in	<i>Format</i>	Format-control string

Return values

<i>None.</i>	

10.22.2.7 API_DebugPrintTrace()

```
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	<i>File</i>	Pointer to a FILE structure
in	<i>RequestID</i>	Pointer to a request id
in	<i>Format</i>	Format-control string

Return values

<i>None.</i>	

10.22.2.8 API_GetServerIDFromObjHandle()

```
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	<i>ObjHandle</i>	input object handle
out	<i>ServerType</i>	server type
out	<i>ID</i>	server id

Return values

<i>0</i>	No error
<i>Other</i>	Value from hpss_LocateRootCS()

10.22.2.9 API_GrabConn()

```
int API_GrabConn (
    const char * FunctionName,
    hpss_reqid_t RequestId,
    int ServerType,
    const hpss_srvr_id_t * ServerID,
    apicsepinfo_t * CSEPinInfo,
    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	<i>FunctionName</i>	calling function name
in	<i>RequestId</i>	request id
in	<i>ServerType</i>	Type of the server
in	<i>ServerID</i>	Server to get connect for
in	<i>CSEPinInfo</i>	core server EP info
out	<i>Conn</i>	connection entry

Return values

<i>-EFAULT</i>	Invalid connection input
<i>-EINVAL</i>	Server's location is not local and cross-cells disabled
<i>-ESYSTEM</i>	Build problem with the library
<i>0</i>	No error
<i>Other</i>	Error returned by hpss_LocateServersByType() or hpss_ConnMgrGrabConn()

10.22.2.10 API_InitRootHandle()

```
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	<i>ThreadContext</i>	thread context
in	<i>RequestID</i>	request id
in, out	<i>HandlePtr</i>	root handle pointer
in, out	<i>FilesetId</i>	root fileset id

Return values

0	Success
<i>HandlePtr</i>	pointer to the NS root handle
<i>Other</i>	Another HPSS errno indicating the nature of the error

10.22.2.11 API_ReleaseConn()

```
int API_ReleaseConn (
    const char * FunctionName,
    hpss_reqid_t RequestId,
    hpss_ConnMgrEntry_t * Conn,
    apicsepinfo_t * CSEPinInfo,
    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 occurred action will be taken to repair the connection before returning control to the caller.

Parameters

in	<i>FunctionName</i>	calling function name
in	<i>RequestId</i>	request id
in	<i>Conn</i>	connection entry
in	<i>CSEPinInfo</i>	core server EP info
in	<i>Error</i>	Error from the server
in	<i>RpcErr</i>	Error from the RPC

Return values

-EBADCONN	The connection went bad
0	No error
<i>Other</i>	Error returned by hpss_ConnMgrReleaseConn()

10.22.2.12 API_SetAcctID()

```
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	<i>AcctID</i>	account ID user mask

Return values

<i>None.</i>	

10.22.2.13 API_SetCurrentGID()

```
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	<i>NewGID</i>	new current Gid

Return values

<i>None.</i>	

10.22.2.14 API_SetCwd()

```
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	<i>CwdPtr</i>	ptr to directory handle
in	<i>AcctCode</i>	ptr to account code
in	<i>COSId</i>	ptr to class of service id
in	<i>FilesetId</i>	Fileset id for cwd
in	<i>Attrs</i>	ptr to directory attributes

This routine will set the current working directory in the global thread state

Return values

<i>None.</i>	

10.22.2.15 API_SetThreadCwd()

```
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	<i>ThreadContext</i>	Thread context to modify
in	<i>ObjHandle</i>	CWD Object Handle
in	<i>FilesetId</i>	Fileset Id for the CWD
in	<i>Account</i>	Account Code for the CWD
in	<i>COSId</i>	COSId for the CWD
in	<i>Attrs</i>	Attributes for CWD

10.22.2.16 API_SetUmask()

```
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.	

10.22.3 Variable Documentation

10.22.3.1 API_TransferType

```
int API_TransferType = 1
```

Mover selects the protocol

10.23 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_ObjHandle_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.

10.23.1 Detailed Description

Functions for manipulating junctions.

10.24 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.

10.24.1 Detailed Description

Functions for creating links.

10.25 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_svr_id_t *ServerId)
Retrieves the ID for a site's root Core Server.

10.25.1 Detailed Description

Functions for looking up object ids or handles.

10.26 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.

10.26.1 Detailed Description

Functions for modifying the offset pointer within a file.

10.27 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 *FTPptr, 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_ilst_t](#) *FildesLocked)
Clear a file descriptor as locked in the list.
- void [API_DescListFree](#) ([hpss_ilst_t](#) *FildesLocked)
Free a descriptor list.
- int [API_DescListIsSet](#) (int Fildes, [hpss_ilst_t](#) *FildesLocked)
Determine whether a file descriptor is locked in the list.
- void [API_DescListSet](#) (int Fildes, [hpss_ilst_t](#) *FildesLocked)
Set a file descriptor as locked in the list.
- [hpss_ilst_t](#) * [API_DescListZero](#) ([hpss_ilst_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_ilst_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_GetNextIORRequestID](#) ([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)
Initilizes a server IOD.
- void [API_InsertInFreeList](#) ([hpss_ilst_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 *FSID`, `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_ilst_t *FildesLocked`, `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_AbortIOByMoverName` (`int32_t SubsystemId`, `char *MoverName`)

Abort all I/O requests for the specified mover name.
- `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_GetNextIORRequestID](#) ()
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_SetNextIORRequestID](#) (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_GetUniqueRequestID or from hpss_ReadQueueAdd.

10.27.1 Detailed Description

Useful misc. Client API functions.

10.27.2 Function Documentation

10.27.2.1 API_AllocateRequestIOR()

```
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 unknown

Return values

<i>other</i>	Error in allocating memory

10.27.2.2 API_BuildFileSrcSink()

```
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	<i>DescList</i>	client IOD descriptor list
in	<i>FTPtr</i>	pointer to the file table
in	<i>OpenBFDescPtr</i>	open bitfile descriptor
in, out	<i>RetDescList</i>	server IOD descriptor list

This routine is called to build the bitfile side for an HPSS IOD source/sink descriptor list.

Return values

<i>0</i>	- on success
<i>-ENOMEM</i>	- if memory allocation fails

10.27.2.3 API_BuildXferSrcSink()

```
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	<i>DescList</i>	Client IOD descriptor list
in	<i>OpenBFDesc</i>	Open bitfile descriptor
in	<i>RequestId</i>	IOD request number
in, out	<i>ListenDesc</i>	Listen socket descriptor
out	<i>DataDesc</i>	Data I/O socket descriptor
out	<i>DataAddrPtr</i>	Data address information
out	<i>MemoryLength</i>	Bytes of memory to use
in, out	<i>RetDescList</i>	Server IOD descriptor list

Return values

0	on success
-ENOMEM	if memory allocation fails
-EIO	if descriptor allocation fails

10.27.2.4 API_CalcBufOffsetAndSize()

```
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	<i>Offset</i>	Client's relative offset into the transfer
in	<i>BufSize</i>	Client's buffer size
in	<i>Buffer</i>	Client's buffer
in	<i>MessageOffset</i>	Message transfer offset
in	<i>MessageLength</i>	Message bytes to transfer
in	<i>RetBufStart</i>	Where in Buffer to start the data transfer
out	<i>RetLength</i>	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`

10.27.2.5 API_CancelDataTask()

```
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	<i>OpenBFDescPtr</i>	Object Handle pointer
in	<i>Flags</i>	Data Thread Flags
in	<i>MutexPtr</i>	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.

10.27.2.6 API_CheckFileDescBounds()

```
int API_CheckFileDescBounds (
    const filetable_t * Ftptr,
    int Fildes )
```

Check the file descriptor against the file table bounds.

Parameters

in	<i>Ftptr</i>	File table
in	<i>Fildes</i>	File descriptor

Return values

<i>HPSS_ERANGE</i>	File descriptor is out of range
<i>HPSS_E_NOERROR</i>	File descriptor is in range

10.27.2.7 API_CleanupInterruptBuffer()

```
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	<i>OpenBFDescPtr</i>	Object Handle pointer

Note

This function assumes that the write descriptor is not being written to while this cleanup is occurring. 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.

10.27.2.8 API_ClearFileTableBusy()

```
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	<i>Ftptr</i>	File table
in	<i>Fildes</i>	Open file descriptor

10.27.2.9 API_CloseDataConnList()

```
void API_CloseDataConnList (
    desc_elem_t ** ListPtr )
```

Closes connections for a file.

Parameters

in	<i>ListPtr</i>	Pointer to the list head

The API_CloseDataConnList routine is called to close the connections for a given file.

10.27.2.10 API_CloseListenDesc()

```
void API_CloseListenDesc (
    int Type,
    int * ListenDesc )
```

Closes a listen port for a thread.

Parameters

in	<i>Type</i>	Type of descriptor
in, out	<i>ListenDesc</i>	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
<i>negative</i>	error closing the descriptor

Note

ListenDesc will be set to API_INVALID_FD after close.

10.27.2.11 API_CloseThrdCntlDesc()

```
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	<i>OpenBFDescPtr</i>	Open Bitfile Descriptor

Return values

0	Success.
<i>Other</i>	Error returned from close()

10.27.2.12 API_ConvertAttrToVAttrSelFlags()

```
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	<i>SelFlags</i>	HPSS selection flags
out	<i>RetFlagsPtr</i>	Ptr to converted flags

Return values

<i>None.</i>	

10.27.2.13 API_ConvertVAttrToAttrs()

```
void API_ConvertVAttrToAttrs (
    const hpss_vattr_t * VAttr,
    hpss_Attrs_t * Attrs )
```

Converts VFS attributes to HPSS attributes.

This routine translates vfs attributes to HPSS attributes.

Parameters

in	<i>VAttr</i>	Ptr to VFS attributes
out	<i>Attrs</i>	Ptr to HPSS attributes

Return values

<i>None</i>	

10.27.2.14 API_ConvertVAttrToAttrSelFlags()

```
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	<i>VAttrSelFlags</i>	NS selection flags
out	<i>RetFlagsPtr</i>	Ptr to converted flags

Return values

<i>None.</i>	

10.27.2.15 API_DescListAlloc()

```
api_desc_node_t* API_DescListAlloc (
    int Fildes )
```

Initialize a descriptor list node.

Parameters

in	<i>Fildes</i>	File descriptor

Returns

A list node containing the provided file descriptor, or NULL

10.27.2.16 API_DescListClear()

```
void API_DescListClear (
    int Fildes,
    hpss_ilist_t * FildesLocked )
```

Clear a file descriptor as locked in the list.

Parameters

in	<i>Fildes</i>	File descriptor
in, out	<i>FildesLocked</i>	Locked file descriptor list

10.27.2.17 API_DescListFree()

```
void API_DescListFree (
    hpss_ilist_t * FildesLocked )
```

Free a descriptor list.

Parameters

in	<i>FildesLocked</i>	Descriptor list to free

10.27.2.18 API_DescListIsSet()

```
int API_DescListIsSet (
    int Fildes,
    hpss_ilist_t * FildesLocked )
```

Determine whether a file descriptor is locked in the list.

Parameters

in	<i>Fildes</i>	File descriptor
in, out	<i>FildesLocked</i>	Locked file descriptor list

10.27.2.19 API_DescListSet()

```
void API_DescListSet (
    int Fildes,
    hpss_ilist_t * FildesLocked )
```

Set a file descriptor as locked in the list.

Parameters

in	<i>Fildes</i>	File descriptor
in, out	<i>FildesLocked</i>	Locked file descriptor list

10.27.2.20 API_DescListZero()

```
hpss_ilst_t* API_DescListZero (
    hpss_ilst_t * FildesLocked )
```

Initialize the descriptor lock list.

Parameters

in, out	<i>FildesLocked</i>	Locked file descriptor list

10.27.2.21 API_DivideFilePath()

```
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	<i>PathIn</i>	path and file
out	<i>PathOut</i>	returned path
out	<i>FileOut</i>	returned file

Return values

<i>-ENAMETOOLONG</i>	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.

10.27.2.22 API_ExpandFileTable()

```
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	<i>Ftptr</i>	File table

Return values

<i>HPSS_E_NOERROR</i>	Expansion completed successfully
<i>HPSS_EMFILE</i>	Too many open files
<i>Other</i>	Other error expanding the open file table

10.27.2.23 API_FileTableSize()

```
uint64_t API_FileTableSize (
    const filetable_t * Ftptr )
```

Get the current file table size.

Parameters

in	<i>Ftptr</i>	File table

Returns

The size of the file table

10.27.2.24 API_FillFreeList()

```
void API_FillFreeList (
    hpss_elist_t * List,
    uint64_t StartDes,
    uint64_t EndDes )
```

Fill entries to the free list in a range [start,end].

Parameters

in	<i>List</i>	List to add to
in	<i>StartDes</i>	Start descriptor number
in	<i>EndDes</i>	End descriptor number

Note

The range includes all integers from StartDes to EndDes-1

10.27.2.25 API_FreeIODPtrs()

```
void API_FreeIODPtrs (
    IOD_t * IODPtr )
```

Frees memory that was allocated to an IOD.

Parameters

in	<i>IODPtr</i>	Pointer to IOD describing request

This routine frees memory that was allocated to the IOD.

10.27.2.26 API_FreeRequestIOR()

```
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	<i>IODPtr</i>	pointer to IOD describing request
in	<i>IORPtr</i>	pointer to IOR to free

Return values

0	Success
-EFAULT	Invalid pointer encountered

10.27.2.27 API_GetAllSubsystems()

```
int API_GetAllSubsystems (
    ls_map_array_t ** ls_map_array )
```

Retrieve a map of all subsystems.

Parameters

out	<i>ls_map_array</i>	Location Map Array

This function finds all subsystems for the current site.

Return values

0	Success
<i>HPSS_ENOENT</i>	No subsystems were returned.
<i>Other</i>	Error communicating with the LS

Note

The caller is responsible for freeing the returned map.

10.27.2.28 API_GetAppliedPath()

```
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	<i>Path</i>	Path
out	<i>Newpath</i>	New Path
in, out	<i>Threadcontext</i>	Thread Context

Return values

0	XQuery does not match any invalid components
<i>-ENOMEM</i>	No memory for new string buffer
<i>Other</i>	Error from API_ApplyRootDir

10.27.2.29 API_GetNextIORequestID()

```
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

10.27.2.30 API_GetObjectFilesetId()

```
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	<i>ThreadContext</i>	Thread Context
in	<i>RequestId</i>	Request Id
in	<i>Ucred</i>	User Credentials
in	<i>ChaseFlags</i>	Flags which determine traversal policy
in	<i>ObjHandle</i>	Object Handle
in	<i>Path</i>	Pathname for lookup
in	<i>CwdStack</i>	Cwd Stack
in	<i>FilesetIdHint</i>	Fileset id hint
in, out	<i>FilesetId</i>	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

10.27.2.31 API_GetOpenTableSlot()

```
uint64_t API_GetOpenTableSlot (
    filetable_t * Ftptr )
```

Find an open file table spot.

Parameters

in	<i>Ftptr</i>	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.

10.27.2.32 API_HexDecode()

```
char* API_HexDecode (
    const char * buffer,
    int * len )
```

Decode a hex buffer as binary into a buffer string.

Parameters

in	<i>buffer</i>	Hex buffer
out	<i>len</i>	Length of the binary buffer

Returns

The buffer, decoded into binary, or NULL.

10.27.2.33 API_HexEncode()

```
char* API_HexEncode (
    const char * buffer,
    int buflen )
```

Encode a digest buffer into a hex string.

Parameters

in	<i>buffer</i>	Digest buffer
in	<i>buflen</i>	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.

10.27.2.34 API_InitializeServerIOD()

```
int API_InitializeServerIOD (
    const hpss_IOD_t * IODPtr,
    IOD_t * RetIODPtr )
```

Intiliizes a server IOD.

This routine is called to initialize an HPSS server IOD.

Parameters

in	<i>IODPtr</i>	client IOD pointer
in, out	<i>RetIODPtr</i>	newly initialize IOD pointer

Return values

0	Success
<i>ENOMEM</i>	If memory allocation fails

10.27.2.35 API_InsertInFreeList()

```
void API_InsertInFreeList (
    hpss_ilist_t * List,
    uint64_t Des )
```

Put an entry back into the free list.

Parameters

in	<i>List</i>	List to add to
in	<i>Des</i>	File descriptor to place back in the table

Note

The file table should be locked.

10.27.2.36 API_InterruptDataTask()

```
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	<i>OpenBFDescPtr</i>	Object Handle pointer

Return values

0	Success.
<i>Other</i>	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.

10.27.2.37 API_LockCondition()

```
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	<i>Lock</i>	Structure for data access control
in	<i>Function</i>	Calling function for logging

Return values

0	Success

10.27.2.38 API_LockMutex()

```
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	<i>MutexPtr</i>	pointer to mutex

Return values

0	- Success

10.27.2.39 API_net_bind()

```
int API_net_bind (
    int sockfd,
    hpss_sockaddr_t * addr,
    char * errbuf,
    size_t errbuflen )
```

Bind wrapper for busy retry handling.

Parameters

in	<i>sockfd</i>	Socket Descriptor
in	<i>addr</i>	Bind Address
in, out	<i>errbuf</i>	Error Buffer
in	<i>errbuflen</i>	Length of error buffer

Note

This function will retry the EADDRINUSE condition using the configured API busy delay and retry values.

10.27.2.40 API_Net_CreateAddr_INADDR_ANY()

```
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	<i>SockAddr</i>	
in	<i>Port</i>	
in	<i>ReqId</i>	

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.

10.27.2.41 API_ObjHandleToString()

```
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	<i>Handle</i>	Object Handle pointer
in	<i>Buffer</i>	Buffer to use

10.27.2.42 API_OpenListenDesc()

```
int API_OpenListenDesc (
    int Type,
    int * ListenDesc,
    data_addr_t * ListenAddr )
```

Opens a list port for a thread.

Parameters

in	<i>Type</i>	Type of descriptor to open
out	<i>ListenDesc</i>	Returned open socket
out	<i>ListenAddr</i>	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
<i>HPSS_ECOMM</i>	Communication error building the socket
<i>HPSS_EINVAL</i>	Type of descriptor

10.27.2.43 API_OpenThrdCntlDesc()

```
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	<i>OpenBFDescPtr</i>	Open Bitfile Descriptor

Return values

0	Success
<i>Other</i>	Error returned from pipe() or fcntl()
<i>EBADF</i>	Input thread control files are in an invalid state

10.27.2.44 API_PerformRetry()

```
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	<i>RetryCB</i>	retry control block
in	<i>Error</i>	error return code

Return values

<i>None.</i>	

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)

10.27.2.45 API_RemoveCachedTrash()

```
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	<i>RequestID</i>	Client Request ID
in	<i>Ucred</i>	User's Credentials
in	<i>FSID</i>	Fileset Id
in	<i>OptionFlags</i>	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

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINVAL</i>	The OptionFlags do not specify whether home or fileset trash should be cleared.

10.27.2.46 API_RetrieveTrashcanHandle()

```
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	<i>ThreadContext</i>	Client API Thread Context
in	<i>RequestID</i>	Client Request ID
in	<i>Ucred</i>	User's Credentials
in	<i>ParentHandle</i>	Parent Object Handle

Parameters

in	<i>FSId</i>	Fileset Id
in	<i>CwdStack</i>	Current Working Directory Stack
in, out	<i>TrashcanContainerHandle</i>	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	<i>TrashcanHandle</i>	This is the actual trashcan directory.
in, out	<i>OptionFlags</i>	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

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_ENOTDIR</i>	The trashcan location is not a directory
<i>Other</i>	Error from hpss_GetPathHandle

10.27.2.47 API_SetFileTableBusy()

```
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	<i>Ftptr</i>	File table
in	<i>Fildes</i>	Open file descriptor
in	<i>FlagChecks</i>	Flags to check before setting the descriptor busy

Return values

<i>0</i>	Success
<i>-EBADF</i>	Not an open bitfile handle, file descriptor out of range, file open with wrong mode
<i>-EBUSY</i>	Some other thread is manipulating the entry
<i>-ESTALE</i>	Connection is no longer valid
<i>-EFAULT</i>	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

10.27.2.48 API_SetLastHPSSErrno()

```
void API_SetLastHPSSErrno (
    apithrdstate_t * Threadcontext,
    int32_t HPSSErrno,
    const char * Func,
    hpss_reqid_t RequestId )
```

Sets the current HPSS errno state.

Parameters

in	<i>Threadcontext</i>	API Thread Context
in	<i>HPSSErrno</i>	HPSS Error Code
in	<i>Func</i>	Name of failing function
in	<i>RequestId</i>	Request Id of failing request

Sets the current HPSS errno state.

10.27.2.49 API_TranslateError()

```
int API_TranslateError (
    int32_t Error,
    retry_cb_t * RetryCB )
```

Translates an HPSS error into a POSIX error.

Parameters

in	<i>Error</i>	error value
in	<i>RetryCB</i>	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.	

10.27.2.50 API_UnlockCondition()

```
int32_t API_UnlockCondition (
    api_condition_variable_t * Lock,
    const char * Function )
```

Unlocks data controlled by a condition variable.

Parameters

in	<i>Lock</i>	Structure for data access control
in	<i>Function</i>	Calling function for logging

Unlock the data protected by the specified Lock structure

Return values

0	Success

10.27.2.51 API_UnlockFileTableEntries()

```
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	<i>SrcSinkPtr</i>	start of src/sink descriptor list
in	<i>FileTabPtr</i>	pointer to current file table
in	<i>FildesLocked</i>	indicates files to be unlocked
in	<i>NumLocked</i>	number of files to unlock
in	<i>LockTable</i>	should we lock/unlock the table?

10.27.2.52 API_UnlockMutex()

```
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	<i>MutexPtr</i>	pointer to mutex

Return values

0	- Success

10.28 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.

10.28.1 Detailed Description

Functions for interfacing with the Core Server's misc. functions.

10.29 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.

10.29.1 Detailed Description

Functions for creating directories.

10.30 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_OpenBitfileVAttr](#) (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.

10.30.1 Detailed Description

Functions for opening HPSS files.

10.31 [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.

10.31.1 Detailed Description

Functions for opening directories.

10.32 [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_attr_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.

10.32.1 Detailed Description

Functions for traversing paths.

10.32.2 Function Documentation

10.32.2.1 API_FreeCwdStack()

```
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	<i>Stack</i>	Pointer to stack

Return values

None.	

10.32.2.2 API_InitCwdStack()

```
int API_InitCwdStack (
    api_cwd_stack_t * Stack,
    uint32_t Local )
```

Initializes a cwd stack.

Parameters

in	<i>Stack</i>	Pointer to cwd stack
in	<i>Local</i>	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.	

10.32.2.3 API_PathFromCwdStack()

```
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	<i>Stack</i>	Pointer to cwd stack
in	<i>Size</i>	Size of memory for path
out	<i>Path</i>	Returned path

Return values

<i>-ENOMEM</i>	if can't allocate memory for stack entry
<i>0</i>	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.

10.32.2.4 API_SetCwdStack()

```
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	<i>CwdStack</i>	Pointer to cwd stack
in	<i>FromStack</i>	Pointer to from stack

Return values

<i>ENOMEM</i>	If can't allocate memory for stack entry
<i>0</i>	otherwise

10.32.2.5 API_TraversePath()

```
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_attr_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	<i>ThreadContext</i>	current context
in	<i>RequestID</i>	request id
in	<i>Ucred</i>	user credentials
in	<i>ObjHandle</i>	object handle
in	<i>Path</i>	pathname
in	<i>CwdStack</i>	cwd stack
in	<i>TraversalFlags</i>	path traversal flags
in	<i>XAttrFlags</i>	extended attrs flags
in	<i>XAttrStoreLevel</i>	extended attrs args
in	<i>SelectFlags</i>	object attributes bits
in	<i>ParentSelectFlags</i>	parent attribute bits
out	<i>RetCwdStack</i>	returned cwd stack
out	<i>RetHandle</i>	object handle
out	<i>RetAttrs</i>	object attributes
out	<i>RetParentHandle</i>	parent handle
out	<i>RetParentAttrs</i>	parent attributes
out	<i>RetObjectName</i>	object name
out	<i>XAttr</i>	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 beyond 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;

```

10.32.2.6 NormalizePath()

```

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	<i>Path</i>	Pointer to path to normalize

Return values

<i>None.</i>	

10.33 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_PIOInitCtx](#) (hpss_pio_t *p)
Start a simplified PIO context.
- int [hpss_PIOReadFile](#) (int hpss_fd, uint64_t data_length, int local_fd, uint64_t *bytes_copied, hpss_read_queue_item_t *item)
Read a file from HPSS to a local file using PIO.
- int [hpss_PIOReadFileMulti](#) (hpss_pio_t *Ctx, int hpss_fd, uint64_t data_length, int local_fd, uint64_t *bytes_copied, hpss_read_queue_item_t *item)
Issue a simplified PIO read.
- 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_PIOResetCtx](#) (hpss_pio_t *p)
Stop a simplified PIO context.
- int [hpss_PIOSetBlockSize](#) (hpss_pio_t *Ctx, uint32_t BlockSize)
Set the client transfer block size.
- int [hpss_PIOSetDefaultCtx](#) (hpss_pio_t *Ctx)

- Set default values in the PIO context.*

 - int [hpss_PIOSetHandleGaps](#) (hpss_pio_t *Ctx, int HandleGaps)

Set the client gap handling behavior.
- int [hpss_PIOSetOffset](#) (hpss_pio_t *Ctx, uint64_t Offset)

Set the offset in the PIO context.
- int [hpss_PIOSetStripeWidth](#) (hpss_pio_t *Ctx, uint32_t Width)

Set the client stripe width.
- int [hpss_PIOSetUseAPIHost](#) (hpss_pio_t *Ctx, int UseAPIHost)

Set the client data hostname behavior.
- int [hpss_PIOStart](#) (hpss_pio_params_t *InputParams, hpss_pio_grp_t *StripeGroup)

Start a new parallel I/O group context.
- int [hpss_PIOWriteFile](#) (int hpss_fd, uint64_t data_length, int local_fd, uint64_t *bytes_copied)

Write a file to HPSS from a local file using PIO.
- int [hpss_PIOWriteFileMulti](#) (hpss_pio_t *Ctx, int hpss_fd, uint64_t data_length, int local_fd, uint64_t *bytes_copied)

Write a file to HPSS from a local file using PIO.
- void [PIOInit](#) (void)

PIOInit.

10.33.1 Detailed Description

Functions for performing Parallel IO.

10.33.2 Function Documentation

10.33.2.1 PIOInit()

```
void PIOInit (
    void )
```

PIOInit.

This function is called to do initialization required by an I/O participant.

10.34 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.

10.35 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.

10.36 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.

10.36.1 Detailed Description

Functions for purging files.

10.37 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.

10.37.1 Detailed Description

Functions for setting the purgelock flag on files.

10.38 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.

10.38.1 Detailed Description

Functions for setting the purgeonmigrate flag on files.

10.39 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_GetdentsAttr](#) (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_ReadAttr](#) (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_ReadAttrHandle](#) (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_ReadAttrPlus](#) (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_ReadRawAttrHandle](#) (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.

10.39.1 Detailed Description

Functions for reading directory entries.

10.39.2 Function Documentation

10.39.2.1 hpss_Rumble()

```
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	<i>SubsystemId</i>	the subsystem for the core server
in	<i>TimeP</i>	return changes after time specified
in	<i>BufferSize</i>	size of output buffer (in bytes)
out	<i>End</i>	hit end of rumble entries
out	<i>RumblePtr</i>	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.

10.40 api_rmlink.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.

10.40.1 Detailed Description

Functions for reading links.

10.41 api_rdlst.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.

10.41.1 Detailed Description

Functions for directly accessing the HPSS readlist feature.

10.42 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.

10.42.1 Detailed Description

Functions for reading HPSS files.

10.43 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.
- u_signed64 [API_AddRegisterValues](#) (uint64_t InitialValue,...)
Add values to a bit register.
- uint64_t [API_RemoveRegisterValues](#) (uint64_t InitialValue,...)
Remove values from a bit register.

10.43.1 Detailed Description

Functions for manipulating bit registers.

10.44 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.

10.44.1 Detailed Description

Functions for renaming files.

10.45 `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.

10.45.1 Detailed Description

Functions for rewinding an open directory.

10.46 `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.

10.46.1 Detailed Description

Functions for removing directories.

10.46.2 Function Documentation

10.46.2.1 Common_hpss_Rmdir()

```
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 to be optionally bypassed.

Parameters

in	<i>Path</i>	path of directory
in	<i>Immediate</i>	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.

10.46.2.2 Common_hpss_RmdirHandle()

```
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	<i>ObjHandle</i>	parent object handle
in	<i>Path</i>	path of directory
in	<i>Ucred</i>	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.

10.47 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 *BFObj, hpss_stage_status_type_t Type, hpss_stage_batch_status_t *Status)
Check the status of files in a batch request.
- int [hpss_GetCallbackStatus](#) (int SubsystemId, hpss_reqid_t CallBackId, hpss_cb_status_t *Status)
Check the status of files in a stage callback 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 *BFObjList, 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.

10.47.1 Detailed Description

Functions for staging data in HPSS.

10.47.2 Function Documentation

10.47.2.1 API_CopyBatchByStatus()

```
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	<i>Batch</i>	Already processed batch
in	<i>Status</i>	Parallel array of statuses for Batch
out	<i>RetryBatch</i>	New batch of retryable entries to retry

10.47.2.2 API_UpdateBatchStatus()

```
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	<i>LocalBatch</i>	Local batch processing potential retries
in	<i>FinalBatch</i>	Final batch containing all requests
out	<i>LocalStatus</i>	Local batch status
out	<i>FinalStatus</i>	Final batch status

10.48 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.

10.48.1 Detailed Description

Functions for retrieving a statistics structure for a file.

10.49 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.

10.49.1 Detailed Description

Functions for retrieving file system statistics.

10.50 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](#) *LimitsOut)
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.

10.50.1 Detailed Description

Functions for retrieving subsystem statistics.

10.51 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.

10.51.1 Detailed Description

Functions for using stream-based I/O.

10.52 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.

10.52.1 Detailed Description

Functions for manipulating symbolic links.

10.53 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.

10.53.1 Detailed Description

Functions for truncating file data.

10.54 `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.

10.54.1 Detailed Description

Functions for manipulating the file creation mask.

10.55 `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.

10.55.1 Detailed Description

Functions for undeleting HPSS files.

10.56 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.

10.56.1 Detailed Description

Functions for unlinking HPSS files.

10.56.2 Function Documentation

10.56.2.1 Common_hpss_Unlink()

```
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	<i>Path</i>	Path of file to unlink
in	<i>Immediate</i>	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.

10.57 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_t *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 which 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.

10.57.1 Detailed Description

Functions for accessing and manipulating User-defined Attributes.

10.58 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.

10.58.1 Detailed Description

Functions for setting the access and modification time of a file.

10.59 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_cstub.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.

10.59.1 Detailed Description

Wrappers around Core Server Interfaces.

10.59.2 Function Documentation

10.59.2.1 API_core_OpenBitfile()

```
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	<i>ThreadContext</i>	Client Thread Context
in	<i>RequestID</i>	Request Id
in	<i>Ucred</i>	User Credentials
in	<i>BitfileObj</i>	HPSS Bitfile Object
in	<i>OpenFlags</i>	Flags
out	<i>RetCOSPtr</i>	COS Values
out	<i>RetSClassPtr</i>	Level 0 Storage Class
out	<i>RetBFHandle</i>	Bitfile Handle

Return values

0	Success.
Other	Error code indicating the nature of the problem

Note

None.

10.60 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.

10.60.1 Detailed Description

Functions for writing file data in HPSS.

10.61 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.

10.61.1 Detailed Description

Functions for accessing the writelist interface.

10.62 gk_RTM.h File Reference

```
#include "rtm_reqlist.h"
```

Classes

- struct [gk_ReqlInfo](#)
Gatekeeper RTM info.

Typedefs

- typedef struct [gk_ReqlInfo](#) [gk_ReqlInfo_t](#)
Gatekeeper RTM info.

10.63 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.

10.64 hacl.h File Reference

```
#include "hpss_api.h"
```


Classes

- struct [hacl_acl_entry](#)
Defines an access control list entry.

Macros

- #define [HACL_M_ALLOW_UNAUTHENTICATED](#) (0x10000000)
Allow unauthenticated 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 /* rwxoid */
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.

10.65 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 "ls_types.h"
#include "lsc_config.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.*

 - struct [api_dist_file_info_t](#)
Distributed file information structure.
 - struct [api_namespec](#)
Name specifier information.
 - struct [api_dist_file_info_t::file_info](#)
Information about the file entry.
 - struct [hpss_errno_state](#)
HPSS Error Code State.
 - struct [hpss_file_hash_digest](#)
Defined the input/output structure for `hpss_FgetFileDigest` and `hpss_FsetFileDigest` and `hpss_FgetFileDigestList` and `hpss_FsetFileDigestList`.
 - struct [hpss_file_hash_digest_list](#)
A list of hash digests for single or multi stripe digest handling.
 - struct [hpss_global_fsent_t](#)
Global fileset entry.
 - struct [hpss_junction_ent_t](#)
HPSS junction entry Structure to hold junction information.
 - struct [hpss_pio_gapinfo_s](#)
PIO sparse file gap information.
 - struct [hpss_pio_params_s](#)
PIO I/O parameters.
 - struct [hpss_userattr](#)
User-defined Attribute pair.
 - struct [hpss_userattr_list](#)
List of User-defined attribute pairs.
 - struct [subsys_stats](#)
HPSS Subsystem statistics.

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`
Referencing 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 Subsystem statistics.

Enumerations

- enum [hpss_file_hash_flags](#) {
[HPSS_FILE_HASH_DIGEST_VALID](#) = (1<<0) , [HPSS_FILE_HASH_SET_BY_USER](#) = (1<<1) ,
[HPSS_FILE_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_NONE](#) , [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) ,
[HPSS_READDIR_RAW](#) = (1 << 1) , [HPSS_READDIR_NFS](#) = (1 << 2) }
- enum [hpss_undelete_flags_t](#) { [HPSS_UNDELETE_NONE](#) = 0 , [HPSS_UNDELETE_RESTORE_TIME](#) ,
[HPSS_UNDELETE_OVERWRITE](#) , [HPSS_UNDELETE_OVERWRITE_AND_RESTORE](#) }
- enum [namespec_type_t](#){ [NAMESPEC_SKIP](#) , [NAMESPEC_REALM](#) , [NAMESPEC_USER](#) , [NAMESPEC_GROUP](#)
 }
- Name specifier translation types.*
- enum [notrunc_flag_t](#) { [NOTRUNC_CLEAR](#) = 0 , [NOTRUNC_SET](#) }
- Truncation flags.*
- enum [purgeunlock_flag_t](#) { [PURGE_UNLOCK](#) = 0 , [PURGE_LOCK](#) , [SUPER_PURGE_UNLOCK](#) ,
[SUPER_PURGE_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_ConvertAttrsToVAttrs](#) (const hpss_Attrs_t *NSAttrs, hpss_vattr_t *VAttrs)
Converts HPSS attributes to VFS attributes.
- 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 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 [hpss_AbortIOByMoverId](#) (int32_t SubsystemId, hpss_srvr_id_t MoverId)

Abort all I/O requests for the specified mover ID.
- int [hpss_AbortIOByMoverName](#) (int32_t SubsystemId, char *MoverName)

Abort all I/O requests for the specified mover name.
- 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_ClearLastHPSS_errno](#) ()
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 *Hints↵
Out, 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](#) *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_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](#) *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](#) SelfFlags, 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](#) SelfFlags, 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](#) SelfFlags, 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_svr_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 *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 *Name, const uint64_t *FilesetId, const ns_ObjHandle_t *FilesetHandle, ns_FilesetAttrBits_t FilesetAttrBitsIn, const ns_FilesetAttrs_t *FilesetAttrIn, ns_FilesetAttrBits_t *FilesetAttrBitsOut, ns_FilesetAttrs_t *FilesetAttrOut)
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](#) *DigestList)
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)
Retrieves 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 *BFObjs, hpss_stage↵
_status_type_t Type, hpss_stage_batch_status_t *Status)
Check the status of files in a batch request.
- int [hpss_GetCallbackStatus](#) (int SubsystemId, hpss_reqid_t CallBackId, hpss_cb_status_t *Status)
Check the status of files in a stage callback 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_GetdentsAttr](#) (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 *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_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_GetJunctionAttr](#) (const char *Path, hpss_Attr_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](#) ()
Retrieves the error state of the last HPSS error.
- int [hpss_GetListAttr](#) (const char *Path, hpss_Attr_t *AttrOut)
Retrieve file statistics associated with directory listing operations.
- hpss_reqid_t [hpss_GetNextIORquestID](#) ()
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_ObjHandle_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.
- int32_t [hpss_LocateRootCS](#) (hpss_reqid_t RequestId, ls_map_t *Location)

Find the local root Core Server.
- int32_t [hpss_LocateServerById](#) (hpss_reqid_t RequestId, hpss_srvr_id_t ServerId, ls_map_t *Location)

Locate a server by its Id This routine locates a server by its unique ID. If ServerId is zero (Invalid), information about the local root Core Server is returned.
- int32_t [hpss_LocateSiteById](#) (hpss_reqid_t RequestId, hpss_id_t SiteId, char *SiteName)

Lookup a site's name by its id.
- int32_t [hpss_LocateSiteByName](#) (hpss_reqid_t RequestId, char *SiteName, hpss_id_t *SiteId)

Lookup a site's id by its name.
- 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_OpenBitfileVAttr](#) (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 *AttrOut)

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_PIOInitCtx](#) ([hpss_pio_t](#) *Ctx)
Start a simplified PIO context.
- int [hpss_PIOReadFile](#) (int hpss_fd, uint64_t data_length, int local_fd, uint64_t *bytes_copied, [hpss_read_queue_item_t](#) *item)
Read a file from HPSS to a local file using PIO.
- int [hpss_PIOReadFileMulti](#) ([hpss_pio_t](#) *Ctx, int hpss_fd, uint64_t data_length, int local_fd, uint64_t *bytes_copied, [hpss_read_queue_item_t](#) *item)
Issue a simplified PIO read.
- 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_PIOResetCtx](#) ([hpss_pio_t](#) *Ctx)
Stop a simplified PIO context.
- int [hpss_PIOSetBlockSize](#) ([hpss_pio_t](#) *Ctx, uint32_t BlockSize)
Set the client transfer block size.
- int [hpss_PIOSetDefaultCtx](#) ([hpss_pio_t](#) *Ctx)
Set default values in the PIO context.
- int [hpss_PIOSetHandleGaps](#) ([hpss_pio_t](#) *Ctx, int HandleGaps)
Set the client gap handling behavior.
- int [hpss_PIOSetStripeWidth](#) ([hpss_pio_t](#) *Ctx, uint32_t Width)
Set the client stripe width.
- int [hpss_PIOSetUseAPIHost](#) ([hpss_pio_t](#) *Ctx, int UseAPIHost)
Set the client data hostname behavior.
- int [hpss_PIOStart](#) ([hpss_pio_params_t](#) *InputParams, [hpss_pio_grp_t](#) *StripeGroup)
Start a new parallel I/O group context.
- int [hpss_PIOWriteFile](#) (int hpss_fd, uint64_t data_length, int local_fd, uint64_t *bytes_copied)
Write a file to HPSS from a local file using PIO.
- int [hpss_PIOWriteFileMulti](#) ([hpss_pio_t](#) *Ctx, int hpss_fd, uint64_t data_length, int local_fd, uint64_t *bytes_copied)
Write a file to HPSS from a local file using PIO.
- signed32 [hpss_PreferedReadOrder](#) (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_ReadAttr](#) (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_ReadAttrHandle](#) (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_ReadAttrPlus](#) (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 *UserCredsP, 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.
- 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_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 SelfFlagsIn, const hpss_vattr_t *AttrIn, uint64_t *SelfFlagsOut, 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_SiteldToName](#) (const hpss_id_t *Siteld, char *SiteName)

Convert Site ID to Site Name.
- int [hpss_SiteNameToId](#) (const char *SiteName, hpss_id_t *Siteld)

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.
- 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](#) *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](#) *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.

10.65.1 Class Documentation

10.65.1.1 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 behavior. O_RDONLY O_WRONLY O_RDWR O_APPEND O_CREAT O_EXCL O_TRUNC

10.66 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.

Typedefs

- typedef struct [hpss_dirent](#) [hpss_dirent_t](#)
HPSS Directory Entry.

10.66.1 Detailed Description

This file contains the definition for the [hpss_dirent](#) structure, which stores HPSS directory name entry information.

10.66.2 Class Documentation

10.66.2.1 struct hpss_dirent

HPSS Directory Entry.

Class Members

ns_ObjHandle_t	d_handle	HPSS Name Server handle
char	d_name[256]	Directory entry name
unsigned16	d_namelen	Length of entry name
u_signed64	d_offset	Offset of next entry
unsigned16	d_reclen	Record length

10.67 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 environment variable.

10.67.1 Detailed Description

Read and Update HPSS Environment variables.

10.67.2 Function Documentation

10.67.2.1 [hpss_GetenvDefault\(\)](#)

```
char* hpss_GetenvDefault (
    char const * Env )
```

Get the default value for an environment 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 representing the variable's default value

10.68 [hpss_Getenv.h](#) File Reference

```
#include <hpss_types.h>
```

Functions

- `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 environment variable.

10.68.1 Function Documentation

10.68.1.1 hpss_GetenvDefault()

```
char* hpss_GetenvDefault (
    char const * Env )
```

Get the default value for an environment 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 representing the variable's default value

10.69 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.

10.69.1 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 Open↔SSL 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.

10.69.2 Macro Definition Documentation

10.69.2.1 ADLER32_BASE

```
#define ADLER32_BASE 65521
```

ADLER32_BASE is the largest prime number smaller than 65536

10.70 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.

10.71 hpss_ihashtable.h File Reference

Generic intrusive hash table.

```
#include <stdbool.h>
#include <stdio.h>
#include "hpss_ilst.h"
```

Typedefs

- typedef `hpss_ilst_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.*

10.71.1 Detailed Description

Generic intrusive hash table.

10.71.2 Typedef Documentation

10.71.2.1 hpss_htable_node_t

```
typedef hpss_ilst_node_t hpss_htable_node_t
```

A hash table node is really just an intrusive list node. But, this is an implementation detail hidden by this typedef.

10.72 hpss_ilst.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_ilst_low_level.h"
```

Typedefs

- typedef [hpss_ilst_t](#) [*\(* hpss_ilst_alloc_func_t\)](#) (hpss_reqid_t RequestID, size_t NumBytes, const char *FileName, const char *FunctionName, const int LineNumber)
Memory allocator for custom list initialization.
- typedef void [\(* hpss_ilst_destructor_t\)](#) ([hpss_ilst_node_t](#) *Node)
Node destructor.
- typedef enum [hpss_ilst_foreach_retval](#) [hpss_ilst_foreach_retval_e](#)
Enumeration for return types of mapped functions.
- typedef struct hpss_ilst_iterator_token * [hpss_ilst_iterator_token_h](#)
Typedef for opaque token.
- typedef [hpss_ilst_foreach_retval_e](#) [\(* hpss_ilst_mapped_func_t\)](#) ([hpss_ilst_node_t](#) *Node)
Mapping function type.
- typedef struct hpss_ilst_node [hpss_ilst_node_t](#)
Just the typedef for the following hpss_ilst_node struct.
- typedef struct hpss_ilst [hpss_ilst_t](#)
List type.

Enumerations

- enum [hpss_ilst_foreach_retval](#)
Enumeration for return types of mapped functions.

Functions

- `int32_t hpss_ilst_append (hpss_ilst_t *List, hpss_ilst_node_t *Node)`
Add a node to the end of an existing list.
- `void hpss_ilst_clear (hpss_ilst_t *List)`
Clear an entire list by taking out all nodes and freeing memory.
- `hpss_ilst_t * hpss_ilst_custom_init (hpss_ilst_destructor_t Destroy, hpss_ilst_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_ilst_empty (hpss_ilst_t *List)`
Check whether a list node points to anything else.
- `void hpss_ilst_foreach (hpss_ilst_t *List, hpss_ilst_mapped_func_t F)`
Map a function across all nodes in a list.
- `void hpss_ilst_foreach_range (hpss_ilst_t *List, hpss_ilst_mapped_func_t F, int32_t StartIndex, int32_t EndIndex)`
Map a function across all nodes in a given range of a list.
- `hpss_ilst_node_t * hpss_ilst_get_current (const hpss_ilst_t *List)`
Return the current node in the list, useful for starting iteration.
- `hpss_ilst_node_t * hpss_ilst_get_current_r (const hpss_ilst_iterator_token_h Token)`
Return the current node in the list, useful for starting reentrant iteration.
- `hpss_ilst_node_t * hpss_ilst_get_node (hpss_ilst_t *List, int32_t Index)`
Get a node at a given index in a list.
- `int32_t hpss_ilst_get_size (const hpss_ilst_t *const List)`
Access the size variable in the private hpss_ilst_t struct.
- `void hpss_ilst_halt_iterator (hpss_ilst_t *List)`
Halt iteration, reset Start, Current, and End to NULL.
- `hpss_ilst_t * hpss_ilst_init (hpss_ilst_destructor_t Destroy)`
Initialize a list.
- `int32_t hpss_ilst_init_iterator (hpss_ilst_t *List)`
Init the iterator to iterate over the entire list.
- `hpss_ilst_iterator_token_h hpss_ilst_init_iterator_r (hpss_ilst_t *List)`
Define a range of indices to include during iteration. Also initialize token for reentrant iteration.
- `int32_t hpss_ilst_init_iterator_range (hpss_ilst_t *List, int32_t Start, int32_t End)`
Init the Current, Start, and End variables in the hpss_ilst struct by giving a range of indices to iterate over.
- `hpss_ilst_iterator_token_h hpss_ilst_init_iterator_range_r (hpss_ilst_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_ilst_insert (hpss_ilst_t *List, hpss_ilst_node_t *Node1, hpss_ilst_node_t *Node2)`
Insert a node into an existing list.
- `int32_t hpss_ilst_join (hpss_ilst_t *List1, hpss_ilst_t *List2)`
Attach one list to the end of another list.
- `int32_t hpss_ilst_prepend (hpss_ilst_t *List, hpss_ilst_node_t *Node)`
Add a node to the start of an existing list.
- `int32_t hpss_ilst_reinit_iterator (hpss_ilst_t *List)`
Reinitialize iteration on the entire list using the halt and initialize functions above.
- `int32_t hpss_ilst_reinit_iterator_range (hpss_ilst_t *List, int32_t Start, int32_t End)`
Reinitialize iteration on a range of indices using the halt and initialize functions above.
- `void hpss_ilst_remove (hpss_ilst_t *List, hpss_ilst_node_t *Node)`
Remove a list node from a list.
- `void hpss_ilst_remove_safe (hpss_ilst_t *List, hpss_ilst_node_t *Node)`
Fail-safe function to remove a list node from a list,.

10.72.1 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_ilst_node_t`. This is the intrusive node within your data structure. You must also create a destructor function that takes an `hpss_ilst_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_ilst` API. The following steps detail how to start using an `hpss_ilst_t`.

1. Build up a list by calling the `hpss_ilst_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_ilst_append`, `hpss_ilst_insert`, `hpss_ilst_insert_index` or `hpss_ilst_prepend` methods.
4. Remove nodes from the list with the `hpss_ilst_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_ilst_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_ilst_node_t`.

The maximum size of an `hpss_ilst_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_ilst_t`: an intrusive linked list structure that is meant to be opaque, meaning devs don't need to manipulate `hpss_ilst_t` member variables to utilize this module
- `node`: an object of type `hpss_ilst_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_ilst_node_t` node and some payload. A dev must create this container struct themselves. If you have an `hpss_ilst_node_t` node and would like to gain access to the container it lives in, use the `hpss_ilst_container` macro.
- `destructor`: a developer defined function that tells the `hpss_ilst` module how to remove nodes from a list. `hpss_ilst_remove` calls the destructor on a node whereas `hpss_ilst_clear` calls the destructor iteratively on every node in the list. Commonly, destructors call `memset` on a node or use `hpss_ilst_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 responsibility to create a struct with payload of your choice
typedef struct myStruct
{
    char *Name; // This is the payload
    hpss_ilst_node_t ListNode; // THIS INTRUSIVE NODE CAN NOT BE A POINTER
}
```

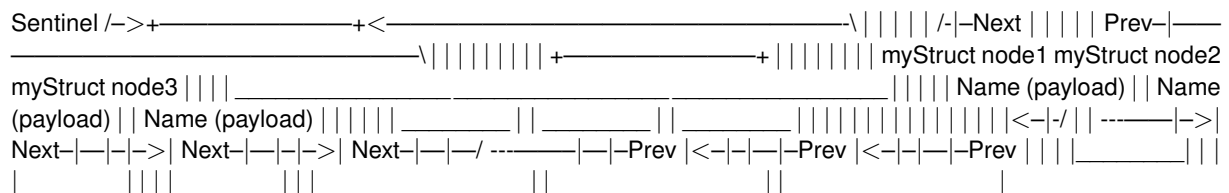
```

} myStruct;
// you need to make your own destructor, but here's a generic example
void my_destructor(hpss_ilst_node_t *NodeToDestroy)
{
    // given the ilist node, get its container and free that
    myStruct *container = hpss_ilst_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_ilst_t *List = hpss_ilst_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_ilst_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_ilst_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_ilst_remove calls my_destructor()
hpss_ilst_remove(List, &newNode1->ListNode);
// remove any remaining node(s) in the list
hpss_ilst_clear(List);

```

Here is what an intrusive linked list with 3 myStruct nodes would look like:



In the above diagram, the outer rectangles are myStruct containers, the inner rectangles are hpss_ilst_node_t nodes, and Name is the payload.

Note

More detailed example file at [^/src/cs/example/ilist_example.c](#)

10.72.2 Typedef Documentation

10.72.2.1 hpss_ilst_foreach_retval_e

```
typedef enum hpss_ilst_foreach_retval hpss_ilst_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_ilst_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.

10.72.3 Enumeration Type Documentation**10.72.3.1 hpss_ilst_foreach_retval**

```
enum hpss_ilst_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_ilst_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.

10.73 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](#) *REnd, [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 Destructer)`
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.

10.73.1 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 $2\log(n+1)$, 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 in worst case.
4. All tree operations (insert, remove, search) maintain strict weak ordering.

10.74 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.
- struct [hpss_irbtree_node](#)
A node in the tree.

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](#) *REnd, [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.

10.74.1 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.

10.75 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.

10.75.1 Detailed Description

Authentication and authorization routines.

10.75.2 Function Documentation

10.75.2.1 hpss_GetAuthType()

```
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	<i>AuthMech</i>	Authentication mechanism - hpss_authn_mech_krb5 (Kerberos) - hpss_authn_mech_unix (Unix)
out	<i>AuthType</i>	Authenticator type hpss_rpc_auth_type_keytab (Keytab) Get the default authenticator type, <i>AuthType</i> , for the specified authentication mechanism, <i>AuthMech</i> .

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_ENOTSUPPORTED</i>	The authentication mechanism is not supported

10.76 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_universaladdressstoport](#) (const char *address)
Return the port from the universal address representation of the address structure.

10.76.1 Detailed Description

HPSS Networking functions for IPV4 and IPV6.

10.76.2 Function Documentation

10.76.2.1 `__hpss_net_maskaddr32()`

```
signed32 __hpss_net_maskaddr32 (
    const hpss_sockaddr_t * addr,
    signed32 mask )
```

Retrieve a 32-bit masked socket address.

Parameters

in	<i>addr</i>	socket address structure
in	<i>mask</i>	mask to use

Deprecated 7.4.1

Note

This function should not be used.

10.76.2.2 `__hpss_net_setscopeid()`

```
int __hpss_net_setscopeid (
    hpss_sockaddr_t * addr,
    char * errbuf,
    size_t errbuflen )
```

Update `sin6_scope_id` for link local addresses.

Parameters

in, out	<i>addr</i>	Socket address structure
out	<i>errbuf</i>	Buffer for more detailed error info
in	<i>errbuflen</i>	Length of <i>errbuf</i>

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.

10.76.2.3 hpss_net_getaddrinfo()

```
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	<i>hostname</i>	Hostname to get the IP address for
in	<i>service</i>	Service to get port for
in	<i>flags</i>	Flags to send to getaddrinfo
in	<i>protocol</i>	Protocol that should be used
out	<i>addr</i>	Socket address structure
out	<i>errbuf</i>	Error buffer for more detailed error info
in	<i>errbuflen</i>	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
<i>EAI_SYSTEM</i>	Check errno for more details, if desired
<i>Other</i>	EAI error code

Note

getaddrinfo() returns non-POSIX error codes in general. In order to decode the error gai_strerror() should be used.

10.76.2.4 hpss_net_getaddrinfo_ex()

```
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	<i>hostname</i>	hostname to get the ip address for
in	<i>service</i>	service to get the port for
in	<i>flags</i>	Flags to send to getaddrinfo
in	<i>protocol</i>	Protocol that should be used
out	<i>addrs</i>	Socket address structure array
out	<i>addrcnt</i>	Count of addresses returned
out	<i>name</i>	Canonical name (if requested)
in	<i>namelen</i>	Storage length of name
out	<i>errbuf</i>	Buffer for more detailed error info
in	<i>errbuflen</i>	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

<i>0</i>	Success
<i>EAI_SYSTEM</i>	Check errno for more detail if desired
<i>Other</i>	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.

10.76.2.5 hpss_net_getsockname()

```
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	<i>sockfd</i>	Socket to get the address from
out	<i>addr</i>	Socket address structure that will contain local information
out	<i>errbuf</i>	Buffer for more detailed error information
in	<i>errbuflen</i>	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

10.77 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_universaladdressstoport` (const char *address)
Return the port from the universal address representation of the address structure.

10.77.1 Function Documentation

10.77.1.1 __hpss_net_maskaddr32()

```
signed32 __hpss_net_maskaddr32 (
    const hpss_sockaddr_t * addr,
    signed32 mask )
```

Retrieve a 32-bit masked socket address.

Parameters

in	<i>addr</i>	socket address structure
in	<i>mask</i>	mask to use

Deprecated 7.4.1

Note

This function should not be used.

10.77.1.2 __hpss_net_setscopeid()

```
int __hpss_net_setscopeid (
    hpss_sockaddr_t * addr,
    char * errbuf,
    size_t errbuflen )
```

Update sin6_scope_id for link local addresses.

Parameters

in, out	<i>addr</i>	Socket address structure
out	<i>errbuf</i>	Buffer for more detailed error info
in	<i>errbuflen</i>	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.

10.77.1.3 hpss_net_getaddrinfo()

```
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	<i>hostname</i>	Hostname to get the IP address for
in	<i>service</i>	Service to get port for
in	<i>flags</i>	Flags to send to getaddrinfo
in	<i>protocol</i>	Protocol that should be used
out	<i>addr</i>	Socket address structure
out	<i>errbuf</i>	Error buffer for more detailed error info
in	<i>errbuflen</i>	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

<i>0</i>	Success
<i>EAI_SYSTEM</i>	Check errno for more details, if desired
<i>Other</i>	EAI error code

Note

`getaddrinfo()` returns non-POSIX error codes in general. In order to decode the error `gai_strerror()` should be used.

10.77.1.4 hpss_net_getaddrinfo_ex()

```
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	<i>hostname</i>	hostname to get the ip address for
in	<i>service</i>	service to get the port for
in	<i>flags</i>	Flags to send to getaddrinfo
in	<i>protocol</i>	Protocol that should be used
out	<i>addrs</i>	Socket address structure array
out	<i>addrcnt</i>	Count of addresses returned
out	<i>name</i>	Canonical name (if requested)
in	<i>namelen</i>	Storage length of name
out	<i>errbuf</i>	Buffer for more detailed error info
in	<i>errbuflen</i>	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

<i>0</i>	Success
<i>EAI_SYSTEM</i>	Check errno for more detail if desired
<i>Other</i>	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.

10.77.1.5 hpss_net_getsockname()

```
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	<i>sockfd</i>	Socket to get the address from
out	<i>addr</i>	Socket address structure that will contain local information
out	<i>errbuf</i>	Buffer for more detailed error information
in	<i>errbuflen</i>	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

10.78 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.

10.78.1 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.

10.79 hpss_stat.h File Reference

```
#include <sys/time.h>
#include "hpss_types.h"
```

Classes

- struct [hpss_stat](#)
Stat structure for HPSS.
- struct [hpss_statfs](#)
File system stats.
- struct [hpss_statfs64](#)
File system stats (64bit)
- struct [hpss_statvfs](#)
Virtual File System stat structure.
- struct [hpss_statvfs64](#)
Virtual File System stat structure (64bit)

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.

10.80 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_il8n.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 of 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 of 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 of 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, char *defString)
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](#) []

10.80.1 Detailed Description

Misc. String Handling Functions

10.80.2 Function Documentation

10.80.2.1 hpss_AdminStateString()

```
const char* hpss_AdminStateString (
    signed32 adminState )
```

Convert the admin state to a string.

Parameters

in	<i>adminState</i>	Admin State

Returns

The admin state as a string, or hpss_Unknown.

10.80.2.2 hpss_AllocStateString()

```
const char* hpss_AllocStateString (
    signed32 allocState )
```

Return a string representation of the given allocation state.

Parameters

in	<i>allocState</i>	Allocation State

Returns

The allocation state as a string, or `hpss_Unknown`.

10.80.2.3 hpss_AuthenticatorTypeFromString()

```
signed32 hpss_AuthenticatorTypeFromString (
    const char * AuthenticatorTypeString,
    hpss_rpc_auth_type_t * AuthenticatorType )
```

Convert the authenticator type string to type.

Parameters

in	<i>AuthenticatorTypeString</i>	Authenticator Type String
in, out	<i>AuthenticatorType</i>	Authenticator Type

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINVAL</i>	Invalid input string

10.80.2.4 hpss_AuthenticatorTypeString()

```
const char* hpss_AuthenticatorTypeString (
    hpss_rpc_auth_type_t AuthenticatorType )
```

Convert the authenticator type to a string.

Parameters

in	<i>AuthenticatorType</i>	Authenticator Type

Returns

The authenticator type as a string, or hpss_Unknown.

10.80.2.5 hpss_AuthnMechTypeFromString()

```
signed32 hpss_AuthnMechTypeFromString (
    const char * AuthnMechString,
    hpss_authn_mech_t * AuthnMech )
```

Convert the authentication mechanism string to type.

Parameters

in	<i>AuthnMechString</i>	Authentication Mechanism String
in, out	<i>AuthnMech</i>	Authentication Mechanism Type

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINVAL</i>	Invalid input string

10.80.2.6 hpss_AuthnMechTypeString()

```
const char* hpss_AuthnMechTypeString (
    hpss_authn_mech_t AuthnMech )
```

Convert the authentication mechanism type to string.

Parameters

in	<i>AuthnMech</i>	Authentication mechanism

Returns

The authentication mechanism type as a string, or hpss_Unknown.

10.80.2.7 hpss_AuthzMechTypeFromString()

```
signed32 hpss_AuthzMechTypeFromString (
    const char * AuthzMechString,
    hpss_authz_mech_t * AuthzMech )
```

Convert the authorization mechanism string to type.

Parameters

in	<i>AuthzMechString</i>	Authorization Mechanism String
in, out	<i>AuthzMech</i>	Authorization Mechanism Type

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINVAL</i>	Invalid input string

10.80.2.8 hpss_AuthzMechTypeString()

```
const char* hpss_AuthzMechTypeString (
    hpss_authz_mech_t AuthzMech )
```

Convert the authorization mechanism type to string.

Parameters

in	<i>AuthzMech</i>	Authorization mechanism

Returns

The authorization mechanism type as a string, or `hpss_Unknown`.

10.80.2.9 hpss_Bytecount32()

```
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	<i>bytecount</i>	Byte Count
in, out	<i>string</i>	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.

10.80.2.10 hpss_Bytecount64()

```
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	<i>bytecount</i>	Byte Count
in, out	<i>string</i>	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.

10.80.2.11 hpss_DriveFlagString()

```
const char* hpss_DriveFlagString (
    signed32 driveFlag )
```

Convert the drive flags to a string.

Parameters

in	<i>driveFlag</i>	Drive Flag

Returns

The drive flag as a string, or hpss_Unknown.

10.80.2.12 hpss_DriveStateString()

```
const char* hpss_DriveStateString (
    signed32 driveState )
```

Convert the drive state to a string.

Parameters

in	<i>driveState</i>	Drive State

Returns

The drive state as a string, or `hpss_Unknown`.

10.80.2.13 hpss_DriveTypeString()

```
const char* hpss_DriveTypeString (
    drive_type_t driveType )
```

Return a string representation of the given drive type.

Parameters

in	<i>driveType</i>	HPSS Drive Type

Returns

The drive type as a string, or "Unknown drive type" if the type cannot be determined

10.80.2.14 hpss_ErrnoName()

```
const char* hpss_ErrnoName (
    signed32 Errno )
```

Convert an error number to a name string.

Parameters

in	<i>Errno</i>	HPSS Error Number

Returns

The defined name of an HPSS error code, or `hpss_Unknown`.

10.80.2.15 hpss_getline()

```
int hpss_getline (
    FILE * fp,
    hpss_stringbuf_t * buf )
```

getline using an hpss conformant string

Parameters

<i>in</i>	<i>fp</i>	Read-open file
<i>in, out</i>	<i>buf</i>	String buffer to read into

Return values

<i>0</i>	something was read (but possibly only a newline)
<i>-1</i>	EOF hit before anything read
<i>other</i>	error

10.80.2.16 hpss_HashTypeString()

```
const char* hpss_HashTypeString (
    hpss_hash_type_t HashType )
```

Convert an error number to a descriptive string.

Parameters

<i>in</i>	<i>Errno</i>	HPSS Error Number

Returns

The string text associated with the error code, or `hpss_Unknown`.

10.80.2.17 hpss_HexString()

```
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

<i>in, out</i>	<i>string</i>	String buffer to use
<i>in</i>	<i>data</i>	Data to stringify
<i>in</i>	<i>len</i>	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 before use

10.80.2.18 hpss_LabelTypeString()

```
const char* hpss_LabelTypeString (
    signed32 labelType )
```

Convert the label type to a string.

Parameters

in	<i>labelType</i>	Label type

Returns

The label type as a string, or `hpss_Unknown`.

10.80.2.19 hpss_MediaTypeString()

```
const char* hpss_MediaTypeString (
    media_type_t mediaType )
```

Return a string representation of the given media type.

Parameters

in	<i>mediaType</i>	HPSS Media Type

Returns

The media type as a string, or "Unknown media type" if the type cannot be determined

10.80.2.20 hpss_MountModeString()

```
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.

10.80.2.21 hpss_OpStateString()

```
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.

10.80.2.22 hpss_ProtLevelFromString()

```
signed32 hpss_ProtLevelFromString (
    const char * ProtLevelString,
    hpss_rpc_prot_level_t * ProtLevel )
```

Convert the protection level string to type.

Parameters

in	<i>ProtLevelString</i>	Protection Level String
in, out	<i>ProtLevel</i>	Protection Level

Return values

<i>HPSS_E_NOERROR</i>	Success
<i>HPSS_EINVAL</i>	Invalid input string

10.80.2.23 hpss_ProtLevelString()

```
const char* hpss_ProtLevelString (
    hpss_rpc_prot_level_t ProtLevel )
```

Convert the protection level to a string.

Parameters

in	<i>ProtLevel</i>	Protection level

Returns

The protection level as a string, or `hpss_Unknown`.

10.80.2.24 hpss_PVLActivityStateString()

```
const char* hpss_PVLActivityStateString (
    signed32 pvlActivityState )
```

Convert the PVL activity state to a string.

Parameters

in	<i>pvlActivityState</i>	PVL activity state

Returns

The PVL activity state as a string, or `hpss_Unknown`.

10.80.2.25 hpss_PVLJobStateString()

```
const char* hpss_PVLJobStateString (
    signed32 pvlJobState )
```

Convert the PVL job state to a string.

Parameters

in	<i>pvlJobState</i>	PVL job state

Returns

The PVL job state as a string, or `hpss_Unknown`.

10.80.2.26 hpss_PVRTypeString()

```
const char* hpss_PVRTypeString (
    unsigned32 PVRType )
```

Convert the pvr type to a string.

Parameters

in	<i>PVRType</i>	PVR type

Returns

The pvr type as a string, or hpss_Unknown.

10.80.2.27 hpss_ServerTypeString()

```
const char* hpss_ServerTypeString (
    unsigned32 ServerType )
```

Convert the server type to a string.

Parameters

in	<i>ServerType</i>	Server type

Returns

The server type as a string, or hpss_Unknown.

10.80.2.28 hpss_string_cmp()

```
int hpss_string_cmp (
    const hpss_string_t * s1,
    const hpss_string_t * s2 )
```

Compare two hpss strings.

Parameters

in	<i>s1</i>	HPSS string
in, out	<i>s2</i>	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

10.80.2.29 hpss_string_fromCstr()

```
void hpss_string_fromCstr (
    hpss_string_t * s,
    const char * str )
```

Create an hpss string from a C String.

Parameters

in	<i>s</i>	C string
in, out	<i>str</i>	HPSS string

10.80.2.30 hpss_stringbuf_clear()

```
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	<i>buf</i>	Conformant string pointer

10.80.2.31 hpss_stringbuf_cmp()

```
int hpss_stringbuf_cmp (
    const hpss_stringbuf_t * buf1,
    const hpss_stringbuf_t * buf2 )
```

Compare two conformant strings.

Parameters

in	<i>buf1</i>	Conformant string pointer
in	<i>buf2</i>	Conformant string pointer

Return values

-1	<i>buf1</i> < <i>buf2</i>
0	<i>buf1</i> == <i>buf2</i>
1	<i>buf1</i> > <i>buf2</i>

10.80.2.32 hpss_stringbuf_delBackward()

```
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	<i>buf</i>	Conformant string pointer
in	<i>cursor</i>	Position inside the buf string
in	<i>num</i>	Amount to delete

Return values

0	Success
<i>other</i>	Error

Note

Cursor updated following this function to point at number bytes prior
This function does not do any bounds checking.

10.80.2.33 hpss_stringbuf_delForward()

```
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

<i>in, out</i>	<i>buf</i>	Conformant string pointer
<i>in</i>	<i>cursor</i>	Position inside the buf string
<i>in</i>	<i>num</i>	Amount to delete

Return values

<i>0</i>	Success
<i>other</i>	Error

Note

Cursor is not updated following this function
 This function does not do any bounds checking.

10.80.2.34 hpss_stringbuf_free()

```
void hpss_stringbuf_free (
    hpss_stringbuf_t * buf )
```

Free a conformant string.

Parameters

<i>in</i>	<i>buf</i>	Conformant string pointer

10.80.2.35 hpss_stringbuf_fromCstr()

```
int hpss_stringbuf_fromCstr (
    hpss_stringbuf_t * buf,
    const char * s,
    size_t len )
```

Create a conformant string from a C string.

Parameters

<i>in, out</i>	<i>buf</i>	Conformant string pointer
<i>in</i>	<i>s</i>	C string
<i>in</i>	<i>len</i>	Length of s

Return values

<i>0</i>	Success
<i>other</i>	Error

10.80.2.36 hpss_stringbuf_fromFile()

```
int hpss_stringbuf_fromFile (
    hpss_stringbuf_t * buf,
    FILE * file )
```

Create a conformant string from a file.

Parameters

<i>in, out</i>	<i>buf</i>	Conformant string pointer
<i>in</i>	<i>file</i>	Open file handle

Return values

<i>0</i>	Success
<i>other</i>	Error

Note

file must be open for reading

10.80.2.37 hpss_stringbuf_fromString()

```
int hpss_stringbuf_fromString (
    hpss_stringbuf_t * buf,
    const hpss_string_t * str )
```

Create a conformant string from an hpss string type.

Parameters

<i>in, out</i>	<i>buf</i>	Conformant string pointer
<i>in</i>	<i>str</i>	HPSS string

Return values

<i>0</i>	Success
<i>other</i>	Error

10.80.2.38 hpss_stringbuf_grow()

```
int hpss_stringbuf_grow (
    hpss_stringbuf_t * buf,
    size_t min )
```

Grow a conformant string.

Parameters

in	<i>buf</i>	Conformant string pointer
in	<i>min</i>	Minimum new size

Return values

<i>ENOMEM</i>	Out of memory
<i>0</i>	Success

Note

No action occurs if the length is already greater than min

10.80.2.39 hpss_stringbuf_insert()

```
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	<i>buf</i>	Conformant string pointer
in, out	<i>cursor</i>	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	<i>str</i>	String to insert

Return values

<i>0</i>	Success
<i>other</i>	Error

Note

Note that no action occurs if str is empty

10.80.2.40 hpss_stringbuf_printf()

```
int hpss_stringbuf_printf (
    hpss_stringbuf_t * buf,
    const char * fmt,
    ... )
```

Write bytes using a format string to a conformant string.

Parameters

in	<i>buf</i>	Conformant string pointer
in	<i>fmt</i>	printf-style format string
in	...	additional parameters to string

Return values

0	Success
<i>other</i>	Error

Note

Note that no action occurs if str is empty

10.80.2.41 hpss_StringClean()

```
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	<i>s</i>	String to clean

10.80.2.42 hpss_StringsAreEqual()

```
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.

10.80.2.43 hpss_time_string()

```
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

10.80.2.44 hpss_Token_Add()

```
int hpss_Token_Add (
    char * token,
    hpss_Tokens_t * tokens )
```

Add a specified token to the array of tokens.

Parameters

in	<i>token</i>	Token to add
in	<i>tokens</i>	Array of tokens

Return values

0	Success
HPSS_ENOMEM	Out of memory

10.80.2.45 hpss_Tokenize()

```
int hpss_Tokenize (
    char * line,
    size_t len,
    hpss_Tokens_t * tokens )
```

Tokenize " or '-delimited C string into HPSS tokens, ignoring spaces.

Parameters

in	<i>line</i>	String to tokenize
in	<i>len</i>	Length of <i>line</i>
in, out	<i>tokens</i>	Tokens retrieved

Return values

0	Success

Note

Tokens cannot include spaces

10.80.2.46 hpss_UsageStateString()

```
const char* hpss_UsageStateString (
    signed32 usageState )
```

Convert the usage state to a string.

Parameters

in	<i>usageState</i>	Usage State

Returns

The usage state as a string, or `hpss_Unknown`.

10.80.2.47 s32_and_map()

```
const char* s32_and_map (
    signed32 s32,
    const s32_str_t * table,
    const unsigned long numEntries,
    char * defString )
```

Find a string entry whose key & the given key returns true.

Parameters

in	<i>s32</i>	Key
in	<i>table</i>	Table to search
in	<i>numEntries</i>	Number of items in the table

Returns

The string found in the table, or "`hpss_Unknown`"

10.80.2.48 s32_eq_map()

```
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	<i>s32</i>	Key
in	<i>table</i>	Table to search
in	<i>numEntries</i>	Number of items in the table

Returns

The string found in the table, or "hpss_Unknown"

10.80.2.49 u32_and_map()

```
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	<i>u32</i>	Key
in	<i>table</i>	Table to search
in	<i>numEntries</i>	Number of items in the table

Returns

The string found in the table, or "hpss_Unknown"

10.80.2.50 u32_eq_map()

```
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	<i>u32</i>	Key
in	<i>table</i>	Table to search
in	<i>numEntries</i>	Number of items in the table

Returns

The string found in the table, or "hpss_Unknown"

10.80.3 Variable Documentation

10.80.3.1 AdminStateStrings

```
const s32_str_t AdminStateStrings[]
```

Initial value:

```
= {
    { 0x00000001 , "Locked"},
    { 0x00000002 , "Unlocked"},
    { 0x00000004 , "Shutdown"},
    { 0x00000008 , "Halt"},
    { 0x00000010 , "Reinitialize"},
    { 0x00000020 , "Repaired"}
}
```

Administration State Strings

10.80.3.2 AuthenticatorTypeStrings

```
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

10.80.3.3 AuthnMechStrings

```
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

10.80.3.4 AuthzMechStrings

```
const u32_str_t AuthzMechStrings[]
```

Initial value:

```
= {
    {hpss_authz_mech_ldap, "ldap"},
    {hpss_authz_mech_unix, "unix"},
}
```

Authorization mechanisms

10.80.3.5 DriveFlagStrings

```
const s32_str_t DriveFlagStrings[]
```

Initial value:

```
= {
    { 0 , "Clear"},
    { 1 , "Modified"},
    { 2 , "PVR/MVR Notify"},
    { 4 , "PVR Notify"},
    { 8 , "MVR Notify"},
    { 16 , "Abort Notify"},
}
```

Drive flag strings

10.80.3.6 DriveStateStrings

```
const s32_str_t DriveStateStrings[]
```

Initial value:

```
= {
    { 0 , "Clear"},
    { 1 , "In use"},
    { 2 , "Free"},
    { 3 , "Dismount Pending"},
    { 4 , "Deleted"},
    { 5 , "Updated"}
}
```

Drive state strings

10.80.3.7 LabelTypeStrings

```
const s32_str_t LabelTypeStrings[]
```

Initial value:

```
= {
    { 0 , "N/A"},
    { 1 , "None"},
    { 2 , "Foreign"},
    { 3 , "HPSS"},
    { 4 , "Data"},
    { 5 , "non-ANSI"},
    { 6 , "NSL"},
    { 7 , "Unknown"},
    { 8 , "Defer (D)"},
    { 9 , "Defer (S)"},
    { 10 , "Defer (O)"}
}
```

Volume label types

10.80.3.8 NumAdminStateStrings

```
const unsigned long NumAdminStateStrings
```

Initial value:

```
=
(sizeof( AdminStateStrings ) / sizeof( s32_str_t ))
```

Number of administration state strings

10.80.3.9 NumAuthenticatorTypeStrings

```
const unsigned long NumAuthenticatorTypeStrings
```

Initial value:

```
=  
(sizeof( AuthenticatorTypeStrings ) / sizeof( u32_str_t ))
```

Number of authenticator type strings

10.80.3.10 NumAuthnMechStrings

```
const unsigned long NumAuthnMechStrings
```

Initial value:

```
=  
(sizeof( AuthnMechStrings ) / sizeof( u32_str_t ))
```

Number of authentication mechanism strings

10.80.3.11 NumAuthzMechStrings

```
const unsigned long NumAuthzMechStrings
```

Initial value:

```
=  
(sizeof( AuthzMechStrings ) / sizeof( u32_str_t ))
```

Number of authorization mechanism strings

10.80.3.12 NumDriveFlagStrings

```
const unsigned long NumDriveFlagStrings
```

Initial value:

```
=  
(sizeof( DriveFlagStrings ) / sizeof( s32_str_t ))
```

Number of drive flag strings

10.80.3.13 NumDriveStateStrings

```
const unsigned long NumDriveStateStrings
```

Initial value:

```
=  
(sizeof( DriveStateStrings ) / sizeof( s32_str_t ))
```

Number of drive state strings

10.80.3.14 NumErrnoStrings

```
const unsigned long NumErrnoStrings
```

Initial value:

```
=  
(sizeof( ErrnoStrings ) / sizeof( u32_str_t ))
```

Number of error strings

10.80.3.15 NumLabelTypeStrings

```
const unsigned long NumLabelTypeStrings
```

Initial value:

```
=  
(sizeof( LabelTypeStrings ) / sizeof( s32_str_t ))
```

Number of volume label type strings

10.80.3.16 NumOpStateStrings

```
const unsigned long NumOpStateStrings
```

Initial value:

```
=  
(sizeof( OpStateStrings ) / sizeof( s32_str_t ))
```

Number of operational state strings

10.80.3.17 NumProtLevelStrings

```
const unsigned long NumProtLevelStrings
```

Initial value:

```
=  
(sizeof( ProtLevelStrings ) / sizeof( u32_str_t ))
```

Number of RPC protection level strings

10.80.3.18 NumPVLActivityStateStrings

```
const unsigned long NumPVLActivityStateStrings
```

Initial value:

```
=  
(sizeof( PVLActivityStateStrings ) / sizeof( s32_str_t ))
```

Number of PVL activity state strings

10.80.3.19 NumPVLJobStateStrings

```
const unsigned long NumPVLJobStateStrings
```

Initial value:

```
=  
(sizeof( PVLJobStateStrings ) / sizeof( s32_str_t ))
```

Number of PVL job state strings

10.80.3.20 NumPVRTypeStrings

```
const unsigned long NumPVRTypeStrings
```

Initial value:

```
=  
(sizeof( PVRTypeStrings ) / sizeof( u32_str_t ))
```

Number of PVR type strings

10.80.3.21 NumServerSubtypeStrings

```
const unsigned long NumServerSubtypeStrings
```

Initial value:

```
=  
(sizeof( ServerSubtypeStrings ) / sizeof( u32_str_t ))
```

Number of server subtype strings

10.80.3.22 NumServerTypeStrings

```
const unsigned long NumServerTypeStrings
```

Initial value:

```
=  
(sizeof( ServerTypeStrings ) / sizeof( u32_str_t ))
```

Number of server strings

10.80.3.23 NumUsageStateStrings

```
const unsigned long NumUsageStateStrings
```

Initial value:

```
=  
(sizeof( UsageStateStrings ) / sizeof( s32_str_t ))
```

Number of usage state strings

10.80.3.24 OpStateStrings

```
const s32_str_t OpStateStrings[]
```

Initial value:

```
= {
    { 0x00000001 , "Enabled"},
    { 0x00000002 , "Disabled"},
    { 0x00000004 , "Suspect"},
    { 0x00000008 , "Minor"},
    { 0x00000010 , "Major"},
    { 0x00000020 , "Broken"}
}
```

Operational States

10.80.3.25 ProtLevelStrings

```
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

10.80.3.26 PVLActivityStateStrings

```
const s32_str_t PVLActivityStateStrings[]
```

Initial value:

```
= {
    { 0 , "Null"},
    { 1 , "Uncommitted"},
    { 2 , "Cartridge Wait"},
    { 3 , "Cartridge Assigned"},
    { 5 , "Drive Wait"},
    { 7 , "Mount Pending"},
    { 8 , "Mount Failed"},
    { 9 , "Mounted"},
    { 10 , "Dismount Pending"},
    { 11 , "Inject"},
    { 12 , "Eject"},
    { 13 , "Reading Label"},
    { 14 , "Unload Pending"},
    { 15 , "Dismounted"},
    { 16 , "Tape Check-In"},
    { 17 , "Tape Check-Out"},
    { 18 , "Tape Shelf Status"}
}
```

PVL activity state strings

10.80.3.27 PVLJobStateStrings

```
const s32_str_t PVLJobStateStrings[]
```

Initial value:

```
= {
  { 0 , "Null"},
  { 1 , "Uncommitted"},
  { 2 , "Cartridge Wait"},
  { 3 , "Drive Wait"},
  { 4 , "Mount Wait"},
  { 5 , "Mounted"},
  { 6 , "Dismount Pending"},
  { 7 , "Aborting"},
  { 8 , "Inject"},
  { 9 , "Eject"},
  { 10 , "In Use"},
  { 11 , "Completed"},
  { 12 , "Deferred Dismount"},
  { 13 , "Tape Check-In"},
  { 14 , "Tape Check-Out"},
  { 15 , "Tape Shelf Status"},
  { 16 , "In Use Reduced"}
}
```

PVL job state strings

10.80.3.28 PVRTYPEStrings

```
const u32_str_t PVRTYPEStrings[]
```

Initial value:

```
= {
  { 3 , "Oper"},
  { 4 , "STK"},
  { 8 , "AML"},
  { 11 , "SCSI"}
}
```

PVR type strings

10.80.3.29 ServerSubtypeStrings

```
const u32_str_t ServerSubtypeStrings[]
```

Initial value:

```
= {
  {0, "none"},
  { 1 , "Disk Storage Server"},
  { 2 , "Tape Storage Server"},
  { 3 , "Operator PVR"},
  { 4 , "STK PVR"},
  { 8 , "AML PVR"},
  { 11 , "SCSI PVR"}
}
```

Server subtypes

10.80.3.30 ServerTypeStrings

```
const u32_str_t ServerTypeStrings[]
```

Initial value:

```
= {
    { 1 , "Core Server"},
    { 3 , "PVL"},
    { 4 , "PVR"},
    { 5 , "Mover"},
    { 6 , "Metadata Monitor"},
    { 9 , "FTP Daemon"},
    { 10 , "PFS Daemon"},
    { 11 , "SSM System Manager"},
    { 13 , "Startup Daemon"},
    { 14 , "Migration/Purge Server"},
    { 21 , "Gatekeeper"},
    { 22 , "RAIT Engine"}
}
```

Server types

10.80.3.31 UsageStateStrings

```
const s32_str_t UsageStateStrings[]
```

Initial value:

```
= {
    { 0x00000001 , "Active"},
    { 0x00000002 , "Idle"},
    { 0x00000004 , "Busy"},
    { 0x00000008 , "Unknown"}
}
```

Usage state strings

10.81 hpss_vattr.h File Reference

HPSS namespace object attributes.

```
#include "hpss_types.h"
#include "hpsoid.h"
#include "hpss_cos.h"
#include "ns_interface_defs.h"
```

10.81.1 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.

10.82 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.

10.82.1 Detailed Description

Functions for creating, manipulating, and parsing XML.

10.83 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.

10.84 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 key↵ OrValue, 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.

10.84.1 Detailed Description

Functions related to parsing structured configuration files.

10.85 hpsscfx_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 "hpsscfx_prototypes.h"
```

Functions

- [hpss_cfg_stanza_t * hpsscfx_LookupHostName](#) ([hpss_cfg_stanza_t](#) *hostStanzaList, const char *host↵ Name, int ignoreDomain)
Find a stanza for a hostname.

10.85.1 Detailed Description

Support functions for working with sections of the HPSS.conf file.

10.86 hpsscfx_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 "hpsscfx_prototypes.h"
#include "hpss_netopt.h"
#include "hpss_errno.h"
#include "hpss_pthread.h"
```

Functions

- int [hpsscfx_ConfGetClientInterfaces](#) (const char *hostName, const char *serverName, int *nwlfCount, hpss_sockaddr_t **nwlfList, char ***nwlfNames)

Lookup client interfaces by host.

10.86.1 Detailed Description

Get Client Interface List from the HPSS.conf file.

10.87 hpsscfx_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 * hpsscfx_ConfParse](#) (char cfgFile[])
Parse the configuration file.
- int [hpsscfx_ConfSetDirPaths](#) (char *theDirString)
Set the Directory Path(s) for the Configuration File.
- int [hpsscfx_ConfSetFileName](#) (char *theFilename)
Set the name of the config file.
- int [hpsscfx_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 [hpsscfx_NetoptGetWriteSize](#) (int SocketDescriptor, hpss_sockaddr_t *IpAddr)
Return the network (TCP/IP) write size to be used for the specified connection.
- int [hpsscfx_NetoptSetSock](#) (int Fd, hpss_sockaddr_t *NetAddress)
Adjust the Socket characteristics.

10.87.1 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.

10.88 hpsscfx_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 [hpsscfx_PatternMatch](#) (char *theString, char *thePattern, int *patternError)
Pattern match function.

10.88.1 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.

10.88.2 Macro Definition Documentation

10.88.2.1 MAX_PATTERN_BUF

```
#define MAX_PATTERN_BUF 1024 /* temporary pattern buffer */
```

Pattern matching executive for braces "{}".

Parameters

in	<i>thePattern</i>	Pattern to be matched
in	<i>theString</i>	candidate string to be matched
out	<i>patternError</i>	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 abcwj abcwkk

If an error is encountered in the pattern string, for example, a missing "}", *patternError is set non-zero on exit.

Return values

<i>TRUE</i>	Pattern match found
<i>FALSE</i>	No pattern match found

Note

1. This function is called recursively.

10.89 hpsscfx_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.

10.89.1 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.

10.90 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)

10.90.1 Macro Definition Documentation

10.90.1.1 CS_DEC64_LEN

```
#define CS_DEC64_LEN 20
```

String length to print 2^{64} as an integer

Index

- __hpss_net_maskaddr32
 - hpss_net.c, [718](#)
 - hpss_net.h, [723](#)
 - __hpss_net_setscopeid
 - hpss_net.c, [718](#)
 - hpss_net.h, [723](#)
 - _internal_hpss_LocateRootCS
 - api_init.c, [609](#)
 - _internal_hpss_LocateServerByID
 - api_init.c, [609](#)
 - _internal_hpss_LocateServersByType
 - api_init.c, [610](#)
- 64-bit Math Library, [496](#)
 - add64_3m, [498](#)
 - add64m, [498](#)
 - and64m, [498](#)
 - andbit64m, [498](#)
 - bld64m, [498](#)
 - cast32m, [498](#)
 - cast64m, [499](#)
 - chkbit64m, [499](#)
 - clrbit64m, [499](#)
 - CONVERT_LONGLONG_TO_U64, [499](#)
 - CONVERT_U64_TO_LONGLONG, [499](#)
 - dec64m, [499](#)
 - div2x64m, [500](#)
 - div64_3m, [500](#)
 - div64m, [500](#)
 - eq64m, [500](#)
 - eqz64m, [500](#)
 - ge64m, [500](#)
 - gt64m, [501](#)
 - high32m, [501](#)
 - inc64m, [501](#)
 - le64m, [501](#)
 - low32m, [501](#)
 - lt64m, [501](#)
 - mem64m, [502](#)
 - mod2x64m, [502](#)
 - mod64_3m, [502](#)
 - mod64m, [502](#)
 - mul64_3m, [502](#)
 - mul64m, [503](#)
 - neq64m, [503](#)
 - neqz64m, [503](#)
 - not64m, [503](#)
 - or64m, [503](#)
 - orbit64m, [503](#)
 - shl64_3m, [504](#)
 - shl64_ipm, [504](#)
 - shl64m, [504](#)
 - shr64_3m, [504](#)
 - shr64_ipm, [504](#)
 - shr64m, [505](#)
 - sub64_3m, [505](#)
 - sub64m, [505](#)
- Account Validation Site Library, [567](#)
 - av_site_AcctIdxToName, [567](#)
 - av_site_AcctNameToIdx, [569](#)
 - av_site_Initialize, [570](#)
 - av_site_Shutdown, [571](#)
 - av_site_ValidateAccount, [571](#)
 - av_site_ValidateChacct, [573](#)
 - av_site_ValidateChown, [574](#)
 - av_site_ValidateCreate, [575](#)
- acct_Site.c, [579](#)
- acct_site.h, [580](#)
- ACLs, [353](#)
 - hpss_DeleteACL, [354](#)
 - hpss_DeleteACLHandle, [355](#)
 - hpss_GetACL, [356](#)
 - hpss_GetACLHandle, [356](#)
 - hpss_SetACL, [357](#)
 - hpss_SetACLHandle, [358](#)
 - hpss_UpdateACL, [359](#)
 - hpss_UpdateACLHandle, [360](#)
- add64_3m
 - 64-bit Math Library, [498](#)
- add64m
 - 64-bit Math Library, [498](#)
- ADLER32_BASE
 - hpss_hash.c, [705](#)
- AdminStateStrings
 - hpss_String.c, [751](#)
- and64m
 - 64-bit Math Library, [498](#)
- andbit64m
 - 64-bit Math Library, [498](#)
- api_access.c, [580](#)
- api_acct.c, [581](#)
 - API_DetermineAcct, [582](#)
 - API_GetAcctForParent, [583](#)
 - API_PutAcctForParent, [583](#)
 - API_SetCurrentSessionAcct, [584](#)
- api_acl.c, [584](#)
- API_AddAllRegisterValues
 - Client API, [95](#)
- API_AddFSDirCache

- api_cache.c, [587](#)
- API_AddHomeDirCache
 - api_cache.c, [587](#)
- API_AddRegisterValues
 - Client API, [95](#)
- API_AllocateRequestIOR
 - api_misc.c, [623](#)
- API_ApplyRootDir
 - api_chroot.c, [595](#)
- API_BuildFileSrcSink
 - api_misc.c, [624](#)
- API_BuildXferSrcSink
 - api_misc.c, [624](#)
- api_cache.c, [585](#)
 - API_AddFSDirCache, [587](#)
 - API_AddHomeDirCache, [587](#)
 - API_ClearFSDirCache, [588](#)
 - API_ClearHomeDirCache, [588](#)
 - API_GetFSDirCache, [589](#)
 - API_GetHomeDirCache, [589](#)
 - API_GetJunctionCache, [590](#)
 - API_InitFSDirCache, [590](#)
 - API_InitHomeDirCache, [590](#)
 - API_RebuildJunctionCache, [591](#)
 - API_RemoveFSDirCache, [591](#)
 - API_RemoveHomeDirCache, [591](#)
 - fsrootkeycompare, [592](#)
 - stale_thresh_cmp_factor, [592](#)
 - thresh_cmp, [593](#)
 - userkeycompare, [593](#)
- API_CalcBlock32
 - Client API, [96](#)
 - Data Structure Conversion, [422](#)
- API_CalcBlock64
 - Client API, [96](#)
 - Data Structure Conversion, [423](#)
- API_CalcBufOffsetAndSize
 - api_misc.c, [625](#)
- API_CancelDataTask
 - api_misc.c, [626](#)
- api_chdir.c, [594](#)
- API_CheckFileDescBounds
 - api_misc.c, [626](#)
- API_ChecksumStateToFlags
 - api_fgetattr.c, [600](#)
- api_chmod.c, [594](#)
- api_chown.c, [595](#)
- api_chroot.c, [595](#)
 - API_ApplyRootDir, [595](#)
- API_CleanupInterruptBuffer
 - api_misc.c, [627](#)
- API_ClearFileTableBusy
 - api_misc.c, [627](#)
- API_ClearFSDirCache
 - api_cache.c, [588](#)
- API_ClearHomeDirCache
 - api_cache.c, [588](#)
- API_ClientAPIInit
 - api_init.c, [610](#)
- api_close.c, [596](#)
- API_CloseDataConnList
 - api_misc.c, [627](#)
- api_closedir.c, [597](#)
- API_CloseListenDesc
 - api_misc.c, [628](#)
- API_CloseThrdCntlDesc
 - api_misc.c, [628](#)
- api_config, [78](#)
- API_CONFIG_ALL_FLAGS
 - Client API, [85](#)
- API_CONFIG_DEFAULT
 - Client API, [85](#)
- API_ConvertAttrsToVAttrs
 - Client API, [97](#)
- API_ConvertAttrToVAttrSelFlags
 - api_misc.c, [629](#)
- API_ConvertModeToPosixMode
 - Client API, [97](#)
 - Data Structure Conversion, [423](#)
- API_ConvertPosixModeToMode
 - Client API, [98](#)
 - Data Structure Conversion, [424](#)
- API_ConvertPosixTimeToTime
 - Client API, [98](#)
 - Data Structure Conversion, [424](#)
- API_ConvertTimeToPosixTime
 - Client API, [99](#)
 - Data Structure Conversion, [425](#)
- API_ConvertVAttrsToAttrs
 - api_misc.c, [629](#)
- API_ConvertVAttrToAttrSelFlags
 - api_misc.c, [630](#)
- API_CopyBatchByStatus
 - api_stage.c, [670](#)
- api_copyfile.c, [597](#)
- API_core_OpenBitfile
 - api_wrappers.c, [679](#)
- api_cos.c, [597](#)
- API_DEBUG_ERROR
 - Client API, [85](#)
- API_DEBUG_NONE
 - Client API, [86](#)
- API_DEBUG_REQUEST
 - Client API, [86](#)
- API_DEBUG_TRACE
 - Client API, [86](#)
- API_DebugPrintError
 - api_init.c, [611](#)
- API_DebugPrintRequest
 - api_init.c, [611](#)
- API_DebugPrintTrace
 - api_init.c, [612](#)
- API_DescListAlloc
 - api_misc.c, [630](#)
- API_DescListClear
 - api_misc.c, [630](#)

API_DescListFree
 api_misc.c, [631](#)
 API_DescListIsSet
 api_misc.c, [631](#)
 API_DescListSet
 api_misc.c, [631](#)
 API_DescListZero
 api_misc.c, [632](#)
 API_DetermineAcct
 api_acct.c, [582](#)
 API_DISABLE_CROSS_REALM
 Client API, [86](#)
 API_DISABLE_JUNCTIONS
 Client API, [86](#)
 api_dist_file_info_t, [80](#)
 api_dist_file_info_t::file_info, [700](#)
 api_distfile.c, [598](#)
 API_DivideFilePath
 api_misc.c, [632](#)
 API_ENABLE_LOGGING
 Client API, [86](#)
 API_ExpandFileTable
 api_misc.c, [632](#)
 api_fgetattr.c, [598](#)
 API_ChecksumStateToFlags, [600](#)
 API_GetMultiFileDigest, [600](#)
 API_GetMultiFileDigestHandle, [601](#)
 API_GetSingleFileDigestHandle, [602](#)
 API_UDAFindKey, [603](#)
 api_fileset.c, [603](#)
 API_FileTableSize
 api_misc.c, [633](#)
 API_FillFreeList
 api_misc.c, [633](#)
 API_FreeCwdStack
 api_path.c, [651](#)
 API_FreelODPtrs
 api_misc.c, [634](#)
 API_FreelIOR
 Client API, [99](#)
 API_FreeRequestIOR
 api_misc.c, [634](#)
 api_fsetattr.c, [604](#)
 API_GetAcctForParent
 api_acct.c, [583](#)
 API_GetAllSubsystems
 api_misc.c, [634](#)
 API_GetAppliedPath
 api_misc.c, [635](#)
 API_GetBatchStatus
 Client API, [99](#)
 API_GetClientAddr
 Client API, [100](#)
 api_getcwd.c, [605](#)
 API_GetDataThreadStackSize
 Client API, [101](#)
 Client API Control, [379](#)
 API_GetFSDirCache
 api_cache.c, [589](#)
 API_GetHomeDirCache
 api_cache.c, [589](#)
 API_GetJunctionCache
 api_cache.c, [590](#)
 API_GetMultiFileDigest
 api_fgetattr.c, [600](#)
 API_GetMultiFileDigestHandle
 api_fgetattr.c, [601](#)
 API_GetNextIORRequestID
 api_misc.c, [635](#)
 API_GetObjectFilesetID
 api_misc.c, [636](#)
 API_GetOpenTableSlot
 api_misc.c, [636](#)
 API_GetServerIDForSubsystem
 Client API, [101](#)
 API_GetServerIDFromObjHandle
 api_init.c, [612](#)
 API_GetSingleFileDigestHandle
 api_fgetattr.c, [602](#)
 api_gettrash.c, [606](#)
 API_GetUniqueRequestID
 Client API, [102](#)
 API_GrabConn
 api_init.c, [613](#)
 API_HexDecode
 api_misc.c, [637](#)
 API_HexEncode
 api_misc.c, [637](#)
 api_ids.c, [606](#)
 api_init.c, [607](#)
 _internal_hpss_LocateRootCS, [609](#)
 _internal_hpss_LocateServerByID, [609](#)
 _internal_hpss_LocateServersByType, [610](#)
 API_ClientAPIInit, [610](#)
 API_DebugPrintError, [611](#)
 API_DebugPrintRequest, [611](#)
 API_DebugPrintTrace, [612](#)
 API_GetServerIDFromObjHandle, [612](#)
 API_GrabConn, [613](#)
 API_InitRootHandle, [613](#)
 API_ReleaseConn, [614](#)
 API_SetAcctID, [614](#)
 API_SetCurrentGID, [615](#)
 API_SetCwd, [615](#)
 API_SetThreadCwd, [616](#)
 API_SetUmask, [616](#)
 API_TransferType, [617](#)
 API_InitCwdStack
 api_path.c, [652](#)
 API_InitFSDirCache
 api_cache.c, [590](#)
 API_InitHomeDirCache
 api_cache.c, [590](#)
 API_InitializeServerIOD
 api_misc.c, [638](#)
 API_InitRootHandle

- api_init.c, 613
- API_InsertInFreeList
 - api_misc.c, 638
- API_InterruptDataTask
 - api_misc.c, 639
- api_junction.c, 617
- api_link.c, 618
- API_LockCondition
 - api_misc.c, 639
- API_LockMutex
 - api_misc.c, 640
- api_lookup.c, 618
- api_lseek.c, 619
- api_misc.c, 619
 - API_AllocateRequestIOR, 623
 - API_BuildFileSrcSink, 624
 - API_BuildXferSrcSink, 624
 - API_CalcBufOffsetAndSize, 625
 - API_CancelDataTask, 626
 - API_CheckFileDescBounds, 626
 - API_CleanupInterruptBuffer, 627
 - API_ClearFileTableBusy, 627
 - API_CloseDataConnList, 627
 - API_CloseListenDesc, 628
 - API_CloseThrdCntlDesc, 628
 - API_ConvertAttrToVAttrSelFlags, 629
 - API_ConvertVAttrToAttrs, 629
 - API_ConvertVAttrToAttrSelFlags, 630
 - API_DescListAlloc, 630
 - API_DescListClear, 630
 - API_DescListFree, 631
 - API_DescListIsSet, 631
 - API_DescListSet, 631
 - API_DescListZero, 632
 - API_DivideFilePath, 632
 - API_ExpandFileTable, 632
 - API_FileTableSize, 633
 - API_FillFreeList, 633
 - API_FreeODPtrs, 634
 - API_FreeRequestIOR, 634
 - API_GetAllSubsystems, 634
 - API_GetAppliedPath, 635
 - API_GetNextIORRequestID, 635
 - API_GetObjectFilesetId, 636
 - API_GetOpenTableSlot, 636
 - API_HexDecode, 637
 - API_HexEncode, 637
 - API_InitializeServerIOD, 638
 - API_InsertInFreeList, 638
 - API_InterruptDataTask, 639
 - API_LockCondition, 639
 - API_LockMutex, 640
 - API_net_bind, 640
 - API_Net_CreateAddr_INADDR_ANY, 640
 - API_ObjHandleToString, 641
 - API_OpenListenDesc, 641
 - API_OpenThrdCntlDesc, 642
 - API_PerformRetry, 642
 - API_RemoveCachedTrash, 643
 - API_RetrieveTrashcanHandle, 644
 - API_SetFileTableBusy, 645
 - API_SetLastHPSSErrno, 646
 - API_TranslateError, 646
 - API_UnlockCondition, 647
 - API_UnlockFileTableEntries, 647
 - API_UnlockMutex, 648
- api_misc_admin.c, 648
- api_mkdir.c, 648
- API_MSGTYPE_MVRPROT
 - Client API, 86
- API_MSGTYPE_PDATA
 - Client API, 86
- api_namespec, 80
- api_namespec_t
 - Client API, 90
- API_net_bind
 - api_misc.c, 640
- API_Net_CreateAddr_INADDR_ANY
 - api_misc.c, 640
- API_ObjHandleToString
 - api_misc.c, 641
- api_open.c, 649
- api_opendir.c, 650
- API_OpenListenDesc
 - api_misc.c, 641
- API_OpenThrdCntlDesc
 - api_misc.c, 642
- api_path.c, 650
 - API_FreeCwdStack, 651
 - API_InitCwdStack, 652
 - API_PathFromCwdStack, 652
 - API_SetCwdStack, 653
 - API_TraversePath, 653
 - NormalizePath, 656
- API_PathFromCwdStack
 - api_path.c, 652
- API_PerformRetry
 - api_misc.c, 642
- api_pio.c, 656
 - PIOInit, 658
- api_posix_lseek.c, 658
- api_posix_open.c, 659
- api_purge.c, 659
- api_purgelock.c, 660
- api_purgeonmigrate.c, 660
- API_PutAcctForParent
 - api_acct.c, 583
- api_rddir.c, 661
 - hpss_Rumble, 662
- api_rddir.c, 662
- api_rddir.c, 663
- api_read.c, 664
- API_RebuildJunctionCache
 - api_cache.c, 591
- api_register.c, 665
- API_ReleaseConn

- api_init.c, [614](#)
- API_RemoveCachedTrash
 - api_misc.c, [643](#)
- API_RemoveFSDirCache
 - api_cache.c, [591](#)
- API_RemoveHomeDirCache
 - api_cache.c, [591](#)
- API_RemoveRegisterValues
 - Client API, [102](#)
- api_rename.c, [665](#)
- API_RetrieveTrashcanHandle
 - api_misc.c, [644](#)
- api_rewdir.c, [666](#)
- api_rmdir.c, [666](#)
 - Common_hpss_Rmdir, [667](#)
 - Common_hpss_RmdirHandle, [667](#)
- API_SetAcctID
 - api_init.c, [614](#)
- API_SetCurrentGID
 - api_init.c, [615](#)
- API_SetCurrentSessionAcct
 - api_acct.c, [584](#)
- API_SetCwd
 - api_init.c, [615](#)
- API_SetCwdStack
 - api_path.c, [653](#)
- API_SetFileTableBusy
 - api_misc.c, [645](#)
- API_SetLastHPSS_errno
 - api_misc.c, [646](#)
- API_SetThreadCwd
 - api_init.c, [616](#)
- API_SetUmask
 - api_init.c, [616](#)
- API_SocketSetOptions
 - Client API Control, [380](#)
- api_stage.c, [669](#)
 - API_CopyBatchByStatus, [670](#)
 - API_UpdateBatchStatus, [671](#)
- API_StageBatchFree
 - Client API, [102](#)
- API_StageBatchGetEntry
 - Client API, [103](#)
- API_StageBatchInit
 - Client API, [103](#)
- API_StageBatchInsertBFHandle
 - Client API, [104](#)
- API_StageBatchInsertBFObj
 - Client API, [105](#)
- API_StageBatchInsertFd
 - Client API, [106](#)
- API_StageStatusFree
 - Client API, [106](#)
- api_stat.c, [671](#)
- api_statfs.c, [672](#)
- api_statsubsys.c, [672](#)
- api_stream.c, [673](#)
- api_symlink.c, [674](#)
- API_TRANSFER_MVRSELECT
 - Client API, [87](#)
- API_TRANSFER_TCP
 - Client API, [87](#)
- API_TransferType
 - api_init.c, [617](#)
- API_TranslateError
 - api_misc.c, [646](#)
- API_TraversePath
 - api_path.c, [653](#)
- api_trunc.c, [674](#)
- API_UDAFindKey
 - api_fgetattr.c, [603](#)
- api_umask.c, [675](#)
- api_undelete.c, [675](#)
- api_unlink.c, [676](#)
 - Common_hpss_Unlink, [676](#)
- API_UnlockCondition
 - api_misc.c, [647](#)
- API_UnlockFileTableEntries
 - api_misc.c, [647](#)
- API_UnlockMutex
 - api_misc.c, [648](#)
- API_UpdateBatchStatus
 - api_stage.c, [671](#)
- API_USE_CONFIG
 - Client API, [87](#)
- API_USE_SAN3P
 - Client API, [87](#)
- api_userattr.c, [677](#)
- api_utime.c, [678](#)
- api_wrappers.c, [679](#)
 - API_core_OpenBitfile, [679](#)
- api_write.c, [680](#)
- api_wrtlist.c, [681](#)
- Authentication, [386](#)
 - hpss_PurgeLoginCred, [387](#)
 - hpss_SetAppLoginCred, [387](#)
 - hpss_SetAuditInfo, [388](#)
 - hpss_SetLoginCred, [389](#)
- AuthenticatorTypeStrings
 - hpss_String.c, [752](#)
- AuthnMechStrings
 - hpss_String.c, [752](#)
- AuthzMechStrings
 - hpss_String.c, [752](#)
- av_site_AcctIdxToName
 - Account Validation Site Library, [567](#)
- av_site_AcctNameToIdx
 - Account Validation Site Library, [569](#)
- av_site_Initialize
 - Account Validation Site Library, [570](#)
- av_site_Shutdown
 - Account Validation Site Library, [571](#)
- av_site_ValidateAccount
 - Account Validation Site Library, [571](#)
- av_site_ValidateChacct
 - Account Validation Site Library, [573](#)

- av_site_ValidateChown
 - Account Validation Site Library, [574](#)
- av_site_ValidateCreate
 - Account Validation Site Library, [575](#)
- bld64m
 - 64-bit Math Library, [498](#)
- Buffered Data, [397](#)
 - hpss_Fclose, [398](#)
 - hpss_Fcntl, [398](#)
 - hpss_Fflush, [399](#)
 - hpss_Fgetc, [400](#)
 - hpss_Fgets, [401](#)
 - hpss_Fopen, [402](#)
 - hpss_Fread, [403](#)
 - hpss_Fseek, [404](#)
 - hpss_Fsync, [405](#)
 - hpss_Ftell, [405](#)
 - hpss_Fwrite, [406](#)
- cast32m
 - 64-bit Math Library, [498](#)
- cast64m
 - 64-bit Math Library, [499](#)
- CFG_COMPOUND_TYPE
 - HPSS Config File API, [544](#)
- CFG_FLAG_TYPE
 - HPSS Config File API, [544](#)
- CFG_SIMPLE_TYPE
 - HPSS Config File API, [544](#)
- chkbit64m
 - 64-bit Math Library, [499](#)
- Client API, [61](#)
 - API_AddAllRegisterValues, [95](#)
 - API_AddRegisterValues, [95](#)
 - API_CalcBlock32, [96](#)
 - API_CalcBlock64, [96](#)
 - API_CONFIG_ALL_FLAGS, [85](#)
 - API_CONFIG_DEFAULT, [85](#)
 - API_ConvertAttrsToVAttrs, [97](#)
 - API_ConvertModeToPosixMode, [97](#)
 - API_ConvertPosixModeToMode, [98](#)
 - API_ConvertPosixTimeToTime, [98](#)
 - API_ConvertTimeToPosixTime, [99](#)
 - API_DEBUG_ERROR, [85](#)
 - API_DEBUG_NONE, [86](#)
 - API_DEBUG_REQUEST, [86](#)
 - API_DEBUG_TRACE, [86](#)
 - API_DISABLE_CROSS_REALM, [86](#)
 - API_DISABLE_JUNCTIONS, [86](#)
 - API_ENABLE_LOGGING, [86](#)
 - API_FreelOR, [99](#)
 - API_GetBatchStatus, [99](#)
 - API_GetClientAddr, [100](#)
 - API_GetDataThreadStackSize, [101](#)
 - API_GetServerIDForSubsystem, [101](#)
 - API_GetUniqueRequestID, [102](#)
 - API_MSGTYPE_MVRPROT, [86](#)
 - API_MSGTYPE_PDATA, [86](#)
 - api_namespec_t, [90](#)
 - API_RemoveRegisterValues, [102](#)
 - API_StageBatchFree, [102](#)
 - API_StageBatchGetEntry, [103](#)
 - API_StageBatchInit, [103](#)
 - API_StageBatchInsertBFHandle, [104](#)
 - API_StageBatchInsertBFObj, [105](#)
 - API_StageBatchInsertFd, [106](#)
 - API_StageStatusFree, [106](#)
 - API_TRANSFER_MVRSELECT, [87](#)
 - API_TRANSFER_TCP, [87](#)
 - API_USE_CONFIG, [87](#)
 - API_USE_SAN3P, [87](#)
 - hpss_AbortIOByMoverId, [107](#)
 - hpss_AbortIOByMoverName, [107](#)
 - hpss_AbortIOByRequestId, [108](#)
 - hpss_AbortIOByRequestMatch, [108](#)
 - hpss_AbortIOByUserId, [110](#)
 - hpss_Access, [111](#)
 - hpss_AccessHandle, [111](#)
 - hpss_AcctCodeToName, [112](#)
 - hpss_AcctNameToCode, [113](#)
 - hpss_BuildLevelString, [113](#)
 - hpss_Chacct, [114](#)
 - hpss_ChacctByName, [115](#)
 - hpss_Chdir, [115](#)
 - hpss_Chmod, [117](#)
 - hpss_Chown, [117](#)
 - hpss_Chroot, [118](#)
 - hpss_ClearLastHPSS_errno, [119](#)
 - hpss_ClientAPIInit, [119](#)
 - hpss_ClientAPIReset, [119](#)
 - hpss_Close, [120](#)
 - hpss_Closedir, [120](#)
 - hpss_ConvertIdsToNames, [121](#)
 - hpss_ConvertNamesToIds, [122](#)
 - hpss_CopyFile, [123](#)
 - hpss_CopyrightStrings, [123](#)
 - hpss_Creat, [123](#)
 - hpss_Create, [124](#)
 - hpss_CreateHandle, [125](#)
 - hpss_DeleteACL, [126](#)
 - hpss_DeleteACLHandle, [127](#)
 - hpss_errno_state_t, [90](#)
 - hpss_Fclear, [128](#)
 - hpss_FclearOffset, [128](#)
 - hpss_Fclose, [130](#)
 - hpss_Fcntl, [131](#)
 - hpss_FdelFileDigestList, [132](#)
 - hpss_Fflush, [132](#)
 - hpss_Fgetc, [133](#)
 - hpss_FgetFileDigest, [134](#)
 - hpss_FgetFileDigestList, [134](#)
 - hpss_Fgets, [135](#)
 - HPSS_FILE, [90](#)
 - HPSS_FILE_HASH_DIGEST_ACTIVE, [92](#)
 - HPSS_FILE_HASH_DIGEST_VALID, [92](#)
 - hpss_file_hash_flags, [91](#)

hpss_file_hash_flags_t, 91
HPSS_FILE_HASH_GENERATED, 92
HPSS_FILE_HASH_MIGRATE_VERIFIED, 92
HPSS_FILE_HASH_SET_BY_USER, 92
HPSS_FILE_HASH_SKIP_VERIFICATION, 92
hpss_file_hash_stripe_flags, 92
hpss_FileGetAttributes, 136
hpss_FileGetAttributesBitfile, 137
hpss_FileGetAttributesHandle, 138
hpss_FileGetDigestHandle, 139
hpss_FileGetDigestListHandle, 139
hpss_FileGetXAttributes, 140
hpss_FileGetXAttributesHandle, 142
hpss_FileSetAttributes, 143
hpss_FileSetAttributesBitfile, 145
hpss_FileSetAttributesHandle, 146
hpss_FileSetCOS, 147
hpss_FileSetCOSHandle, 148
hpss_FilesetCreate, 149
hpss_FilesetDelete, 150
hpss_FilesetGetAttributes, 150
hpss_FilesetListAll, 151
hpss_FilesetSetAttributes, 152
hpss_Fopen, 152
hpss_Fpreallocate, 154
hpss_Fread, 154
hpss_Fseek, 155
hpss_FsetFileDigest, 156
hpss_FsetFileDigestList, 157
hpss_Fstat, 158
hpss_Fsync, 159
hpss_Ftell, 159
hpss_Ftruncate, 159
hpss_Fwrite, 160
hpss_GetAcct, 161
hpss_GetAcctName, 162
hpss_GetACL, 162
hpss_GetACLHandle, 163
hpss_GetAllSubsystems, 164
hpss_GetAsyncStatus, 164
hpss_GetAttrHandle, 165
hpss_GetBatchAsynchStatus, 166
hpss_GetCallbackStatus, 166
hpss_GetConfiguration, 167
hpss_Getcwd, 167
hpss_Getdents, 168
hpss_GetdentsAttrs, 168
hpss_GetDistFile, 169
hpss_GetFileNotrunc, 170
hpss_GetFullPath, 170
hpss_GetFullPathBitfile, 171
hpss_GetFullPathBitfileLen, 171
hpss_GetFullPathHandle, 172
hpss_GetFullPathHandleLen, 173
hpss_GetFullPathLen, 173
hpss_GetGID, 174
hpss_GetIntersectedList, 174
hpss_GetJunctionAttrs, 175
hpss_GetJunctions, 175
hpss_GetLastHPSSErrno, 176
hpss_GetListAttrs, 176
hpss_GetNextIORequestID, 177
hpss_GetObjFromHandle, 177
hpss_GetObjType, 178
hpss_GetPathHandle, 178
hpss_GetPathHandleObjHash, 179
hpss_GetPathObjHash, 179
hpss_GetRawAttrHandle, 180
hpss_GetSubSysLimits, 180
hpss_GetSubSysStats, 181
hpss_GetThreadUcred, 181
hpss_GetTrashcan, 182
hpss_GetTrashcanHandle, 183
hpss_GetTrashSettings, 184
hpss_GetUID, 184
HPSS_HASH_SINGLE, 92
HPSS_HASH_SINGLE_ONLY, 92
HPSS_HASH_STRIPE, 92
HPSS_HASH_STRIPE_ONLY, 92
hpss_HashFlagsToString, 184
hpss_InsertDistFile, 185
hpss_JunctionCreate, 185
hpss_JunctionCreateHandle, 186
hpss_JunctionDelete, 187
hpss_JunctionDeleteHandle, 187
hpss_Link, 188
hpss_LinkHandle, 189
hpss_LinkHandleParent, 189
hpss_LoadDefaultThreadState, 190
hpss_LoadThreadState, 191
hpss_LocateRootCS, 192
hpss_LocateServerByID, 192
hpss_LocateSiteByld, 193
hpss_LocateSiteByName, 193
hpss_LookupRootCS, 194
hpss_Lseek, 194
hpss_Lstat, 195
hpss_Migrate, 196
HPSS_MIGRATE_FORCE, 87
HPSS_MIGRATE_HPSS_ONLY, 87
HPSS_MIGRATE_NO_COPY, 87
HPSS_MIGRATE_PURGE_DATA, 87
hpss_MiscAdmin, 196
hpss_Mkdir, 197
hpss_MkdirHandle, 198
HPSS_O_EXCL, 88
HPSS_O_NO_TAPE, 88
HPSS_O_STAGE_ASYNC, 88
HPSS_O_STAGE_BKGRD, 88
HPSS_O_STAGE_NONE, 88
hpss_Open, 199
hpss_OpenBitfile, 200
hpss_OpenBitfileVAttrs, 202
hpss_Opendir, 203
hpss_OpendirHandle, 204
hpss_OpenHandle, 205

- hpss_PingCore, 206
- HPSS_PIO_HANDLE_GAP, 88
- HPSS_PIO_MVR_SELECT, 93
- HPSS_PIO_NONE, 92
- hpss_pio_operation_t, 92
- hpss_pio_options_t, 91
- HPSS_PIO_PORT_RANGE, 88
- HPSS_PIO_PUSH, 89
- HPSS_PIO_READ, 92
- HPSS_PIO_TCPIP, 93
- hpss_pio_transport_t, 92
- HPSS_PIO_USE_API_HOSTNAME, 89
- HPSS_PIO_WRITE, 92
- hpss_PIOEnd, 207
- hpss_PIOExecute, 207
- hpss_PIOExecuteItem, 208
- hpss_PIOExportGrp, 209
- hpss_PIOFinalize, 210
- hpss_PIOGetPartError, 210
- hpss_PIOGetRequestID, 211
- hpss_PIOImportGrp, 211
- hpss_PIOInitCtx, 212
- hpss_PIOReadFile, 212
- hpss_PIOReadFileMulti, 213
- hpss_PIORegister, 214
- hpss_PIOResetCtx, 214
- hpss_PIOSetBlockSize, 215
- hpss_PIOSetDefaultCtx, 215
- hpss_PIOSetHandleGaps, 216
- hpss_PIOSetStripeWidth, 216
- hpss_PIOSetUseAPIHost, 217
- hpss_PIOStart, 217
- hpss_PIOWriteFile, 218
- hpss_PIOWriteFileMulti, 218
- hpss_PreferedReadOrder, 219
- hpss_Purge, 220
- hpss_PurgeLock, 220
- hpss_PurgeLoginCred, 221
- hpss_PurgeOnMigrate, 222
- hpss_Read, 222
- hpss_ReadAttrs, 223
- hpss_ReadAttrsHandle, 224
- hpss_ReadAttrsPlus, 224
- hpss_Readdir, 225
- hpss_readdir_flags_t, 93
- HPSS_READDIR_GETATTRS, 93
- HPSS_READDIR_NFS, 93
- HPSS_READDIR_NONE, 93
- HPSS_READDIR_RAW, 93
- hpss_ReaddirHandle, 226
- hpss_ReaddirPlus, 227
- hpss_ReadItem, 228
- hpss_Readlink, 228
- hpss_ReadlinkHandle, 229
- hpss_ReadList, 230
- hpss_ReadQueueAdd, 230
- hpss_ReadQueueAddMany, 231
- hpss_ReadQueueCreateContext, 231
- hpss_ReadQueueList, 232
- hpss_ReadQueueRemove, 232
- hpss_ReadQueueRemoveContext, 233
- hpss_ReadRawAttrsHandle, 233
- hpss_RemoveIntersectionFromList, 234
- hpss_Rename, 235
- hpss_RenameHandle, 236
- hpss_ReopenBitfile, 237
- hpss_ResetSubSysStats, 238
- hpss_Rewinddir, 238
- hpss_Rmdir, 239
- hpss_RmdirHandle, 239
- hpss_RmdirHandleImmediate, 240
- hpss_RmdirImmediate, 241
- hpss_SetAcct, 242
- hpss_SetAcctByName, 242
- hpss_SetACL, 243
- hpss_SetACLHandle, 243
- hpss_SetAPILogLevel, 244
- hpss_SetAPILogPath, 245
- hpss_SetApplicationName, 245
- hpss_SetAppLoginCred, 246
- hpss_SetAttrHandle, 246
- hpss_SetAuditInfo, 247
- hpss_SetConfiguration, 248
- hpss_SetCOSByHints, 248
- hpss_SetFileNotrunc, 249
- hpss_SetFileOffset, 250
- hpss_SetGID, 250
- hpss_SetLoginCred, 251
- hpss_SetNextIORquestID, 252
- hpss_SiteldToName, 252
- hpss_SiteNameTold, 253
- hpss_Stage, 253
- HPSS_STAGE_STATUS_ACTIVE, 89
- HPSS_STAGE_STATUS_QUEUED, 89
- HPSS_STAGE_STATUS_UNKNOWN, 89
- hpss_StageBatch, 254
- hpss_StageBatchCallBack, 255
- hpss_StageCallBack, 255
- hpss_StageCallBackBitfile, 257
- hpss_Stat, 258
- hpss_Statfs, 258
- hpss_Statfs64, 259
- hpss_Statvfs, 259
- hpss_Statvfs64, 260
- hpss_Symlink, 260
- hpss_SymlinkHandle, 261
- hpss_ThreadCleanUp, 262
- hpss_Truncate, 262
- hpss_TruncateHandle, 263
- hpss_Umask, 264
- hpss_Undelete, 264
- hpss_undelete_flags_t, 93
- HPSS_UNDELETE_NONE, 93
- HPSS_UNDELETE_OVERWRITE, 93
- HPSS_UNDELETE_OVERWRITE_AND_RESTORE, 93

- HPSS_UNDELETE_RESTORE_TIME, 93
- hpss_UndeleteHandle, 265
- hpss_Unlink, 265
- hpss_UnlinkHandle, 266
- hpss_UnlinkHandleImmediate, 267
- hpss_UnlinkImmediate, 268
- hpss_UpdateACL, 268
- hpss_UpdateACLHandle, 269
- hpss_userattr_t, 91
- hpss_UserAttrCloseCursor, 270
- hpss_UserAttrDeleteAttrHandle, 270
- hpss_UserAttrDeleteAttrs, 271
- hpss_UserAttrGetAttrHandle, 272
- hpss_UserAttrGetAttrs, 273
- hpss_UserAttrListAttrHandle, 273
- hpss_UserAttrListAttrs, 274
- hpss_UserAttrReadObj, 275
- hpss_UserAttrReadXML, 276
- hpss_UserAttrReadXMLObj, 276
- hpss_UserAttrSetAttrHandle, 277
- hpss_UserAttrSetAttrs, 278
- hpss_UserAttrXQueryGet, 278
- hpss_UserAttrXQueryGetHandle, 279
- hpss_UserAttrXQueryUpdate, 280
- hpss_UserAttrXQueryUpdateHandle, 280
- hpss_Utime, 281
- hpss_Write, 282
- hpss_WriteList, 282
- MAX_XPATH_ELEM_SIZE, 89
- MAX_XPATH_SIZE, 89
- NAMESPEC_GROUP, 93
- NAMESPEC_REALM, 93
- NAMESPEC_SKIP, 93
- namespec_type_t, 93
- NAMESPEC_USER, 93
- NOTRUNC_CLEAR, 94
- notrunc_flag_t, 94
- NOTRUNC_SET, 94
- PURGE_LOCK, 94
- PURGE_ON_MIGRATE_CLEAR, 95
- PURGE_ON_MIGRATE_SET, 95
- PURGE_UNLOCK, 94
- purgelock_flag_t, 94
- purgeonmigrate_flag_t, 94
- SUPER_PURGE_LOCK, 94
- SUPER_PURGE_UNLOCK, 94
- UDA_API_VALUE, 89
- UDA_API_XML, 90
- XML_ATTR, 90
- XML_NO_ATTR, 90
- Client API Control, 379
 - API_GetDataThreadStackSize, 379
 - API_SocketSetOptions, 380
 - hpss_ClientAPIInit, 380
 - hpss_ClientAPIReset, 382
 - hpss_GetConfiguration, 382
 - hpss_LoadDefaultThreadState, 382
 - hpss_MiscAdmin, 383
 - hpss_SetAPILogLevel, 384
 - hpss_SetAPILogPath, 385
 - hpss_SetApplicationName, 385
 - hpss_SetConfiguration, 386
 - hpss_ThreadCleanUp, 386
- Client API Location Services, 524
 - hpss_LocateRootCS, 524
 - hpss_LocateServerByID, 525
 - hpss_LocateServersByType, 525
 - hpss_LocateSiteById, 526
 - hpss_LocateSiteByName, 527
- clrbt64m
 - 64-bit Math Library, 499
- Common_hpss_Rmdir
 - api_rmdir.c, 667
- Common_hpss_RmdirHandle
 - api_rmdir.c, 667
- Common_hpss_Unlink
 - api_unlink.c, 676
- Common_hpss_UnlinkHandle
 - File Open, Create, and Close, 284
- CONVERT_LONGLONG_TO_U64
 - 64-bit Math Library, 499
- CONVERT_U64_TO_LONGLONG
 - 64-bit Math Library, 499
- CS_DEC64_LEN
 - u_signed64.h, 767
- Data Containers, 445
 - DEFAULT_HASHTABLE_SIZE, 454
 - hpss_ihashtable_constants, 453
 - hpss_ilst_alloc_func_t, 451
 - hpss_ilst_append, 454
 - hpss_ilst_clear, 454
 - hpss_ilst_custom_init, 455
 - hpss_ilst_destructor_t, 451
 - hpss_ilst_empty, 455
 - hpss_ilst_foreach, 456
 - hpss_ilst_foreach_range, 456
 - hpss_ilst_get_current, 457
 - hpss_ilst_get_current_r, 457
 - hpss_ilst_get_node, 458
 - hpss_ilst_get_size, 458
 - hpss_ilst_halt_iterator, 459
 - hpss_ilst_init, 459
 - hpss_ilst_init_iterator, 459
 - hpss_ilst_init_iterator_r, 460
 - hpss_ilst_init_iterator_range, 460
 - hpss_ilst_init_iterator_range_r, 461
 - hpss_ilst_insert, 461
 - hpss_ilst_iterator_token_h, 451
 - hpss_ilst_join, 462
 - hpss_ilst_mapped_func_t, 451
 - hpss_ilst_prepend, 462
 - hpss_ilst_reinit_iterator, 463
 - hpss_ilst_reinit_iterator_range, 463
 - hpss_ilst_remove, 464
 - hpss_ilst_remove_safe, 464
 - hpss_ilst_t, 452

- hpss_irbtree_clear, [465](#)
- hpss_irbtree_color, [454](#)
- hpss_irbtree_container, [450](#)
- hpss_irbtree_empty, [465](#)
- hpss_irbtree_find, [465](#)
- hpss_irbtree_find_cmp, [466](#)
- hpss_irbtree_foreach, [467](#)
- hpss_irbtree_foreach_node, [467](#)
- hpss_irbtree_foreach_node_prev, [468](#)
- hpss_irbtree_init, [468](#)
- hpss_irbtree_insert, [469](#)
- hpss_irbtree_insert_multi, [469](#)
- hpss_irbtree_lower_bound, [469](#)
- hpss_irbtree_max, [470](#)
- hpss_irbtree_min, [470](#)
- hpss_irbtree_node_callback_t, [452](#)
- hpss_irbtree_node_comparator_t, [452](#)
- hpss_irbtree_node_destructor_t, [453](#)
- hpss_irbtree_node_next, [471](#)
- hpss_irbtree_node_prev, [471](#)
- hpss_irbtree_node_printer_t, [453](#)
- hpss_irbtree_print, [471](#)
- hpss_irbtree_remove, [472](#)
- hpss_irbtree_size, [472](#)
- hpss_irbtree_upper_bound, [473](#)
- IRBTREE_BLACK, [454](#)
- IRBTREE_RED, [454](#)
- MAX_LOAD_FACTOR, [454](#)
- Data Structure Conversion, [422](#)
 - API_CalcBlock32, [422](#)
 - API_CalcBlock64, [423](#)
 - API_ConvertModeToPosixMode, [423](#)
 - API_ConvertPosixModeToMode, [424](#)
 - API_ConvertPosixTimeToTime, [424](#)
 - API_ConvertTimeToPosixTime, [425](#)
- dec64m
 - 64-bit Math Library, [499](#)
- DEFAULT_HASHTABLE_SIZE
 - Data Containers, [454](#)
- Directory Creation and Deletion, [360](#)
 - hpss_Mkdir, [361](#)
 - hpss_MkdirHandle, [362](#)
 - hpss_Rmdir, [362](#)
 - hpss_RmdirHandle, [363](#)
 - hpss_RmdirHandleImmediate, [364](#)
 - hpss_RmdirImmediate, [365](#)
- Directory Data, [365](#)
 - hpss_Closedir, [366](#)
 - hpss_Getdents, [367](#)
 - hpss_GetdentsAttrs, [368](#)
 - hpss_Opendir, [369](#)
 - hpss_OpendirHandle, [369](#)
 - hpss_ReadAttrs, [370](#)
 - hpss_ReadAttrsHandle, [370](#)
 - hpss_ReadAttrsPlus, [371](#)
 - hpss_Readdir, [372](#)
 - hpss_ReaddirHandle, [373](#)
 - hpss_ReaddirPlus, [374](#)
- hpss_ReadRawAttrsHandle, [375](#)
- hpss_Rewinddir, [375](#)
- div2x64m
 - 64-bit Math Library, [500](#)
- div64_3m
 - 64-bit Math Library, [500](#)
- div64m
 - 64-bit Math Library, [500](#)
- DriveFlagStrings
 - hpss_String.c, [752](#)
- DriveStateStrings
 - hpss_String.c, [753](#)
- eq64m
 - 64-bit Math Library, [500](#)
- eqz64m
 - 64-bit Math Library, [500](#)
- File Attribute, [322](#)
 - hpss_Access, [324](#)
 - hpss_AccessHandle, [324](#)
 - hpss_Chacct, [325](#)
 - hpss_ChacctByName, [326](#)
 - hpss_Chmod, [326](#)
 - hpss_Chown, [327](#)
 - hpss_FileGetAttributes, [327](#)
 - hpss_FileGetAttributesBitfile, [328](#)
 - hpss_FileGetAttributesHandle, [329](#)
 - hpss_FileGetXAttributes, [330](#)
 - hpss_FileGetXAttributesHandle, [331](#)
 - hpss_FileSetAttributes, [332](#)
 - hpss_FileSetAttributesBitfile, [334](#)
 - hpss_FileSetAttributesHandle, [335](#)
 - hpss_FileSetCOS, [336](#)
 - hpss_FileSetCOSHandle, [337](#)
 - hpss_Fstat, [338](#)
 - hpss_GetAttrHandle, [338](#)
 - hpss_GetFileNotrunc, [339](#)
 - hpss_GetListAttrs, [339](#)
 - hpss_GetRawAttrHandle, [340](#)
 - hpss_Lstat, [341](#)
 - hpss_PurgeLock, [341](#)
 - hpss_PurgeOnMigrate, [342](#)
 - hpss_SetAttrHandle, [343](#)
 - hpss_SetCOSByHints, [344](#)
 - hpss_SetFileNotrunc, [344](#)
 - hpss_Stat, [345](#)
 - hpss_Utime, [345](#)
- File Data, [298](#)
 - hpss_CopyFile, [300](#)
 - hpss_Fclear, [300](#)
 - hpss_FclearOffset, [301](#)
 - hpss_Fpreallocate, [301](#)
 - hpss_Ftruncate, [302](#)
 - hpss_Lseek, [303](#)
 - hpss_PREFERREDReadOrder, [303](#)
 - hpss_Read, [305](#)
 - hpss_ReadItem, [306](#)
 - hpss_ReadQueueAdd, [306](#)

- hpss_ReadQueueAddMany, [307](#)
- hpss_ReadQueueCreateContext, [308](#)
- hpss_ReadQueueList, [308](#)
- hpss_ReadQueueRemove, [309](#)
- hpss_ReadQueueRemoveContext, [309](#)
- hpss_SetFileOffset, [310](#)
- hpss_SetNextIORequestID, [310](#)
- hpss_Truncate, [311](#)
- hpss_TruncateHandle, [312](#)
- hpss_Write, [312](#)
- File Name, [346](#)
 - hpss_Link, [347](#)
 - hpss_LinkHandle, [347](#)
 - hpss_LinkHandleParent, [349](#)
 - hpss_Rename, [350](#)
 - hpss_RenameHandle, [351](#)
 - hpss_Symlink, [352](#)
 - hpss_SymlinkHandle, [352](#)
- File Open, Create, and Close, [283](#)
 - Common_hpss_UnlinkHandle, [284](#)
 - hpss_Close, [285](#)
 - hpss_Creat, [285](#)
 - hpss_Create, [286](#)
 - hpss_CreateHandle, [287](#)
 - hpss_Open, [288](#)
 - hpss_OpenBitfile, [289](#)
 - hpss_OpenBitfileVAttr, [290](#)
 - hpss_OpenHandle, [291](#)
 - hpss_ReopenBitfile, [293](#)
 - hpss_Unlink, [294](#)
 - hpss_UnlinkHandle, [296](#)
 - hpss_UnlinkHandleImmediate, [297](#)
 - hpss_UnlinkImmediate, [298](#)
- Fileset and Junction, [313](#)
 - hpss_FilesetCreate, [314](#)
 - hpss_FilesetDelete, [315](#)
 - hpss_FilesetGetAttributes, [316](#)
 - hpss_FilesetListAll, [317](#)
 - hpss_FilesetSetAttributes, [317](#)
 - hpss_GetJunctionAttr, [318](#)
 - hpss_GetJunctions, [319](#)
 - hpss_JunctionCreate, [319](#)
 - hpss_JunctionCreateHandle, [320](#)
 - hpss_JunctionDelete, [321](#)
 - hpss_JunctionDeleteHandle, [321](#)
- fsrootkeycompare
 - api_cache.c, [592](#)
- Gatekeeper Site Library, [557](#)
 - gk_EntryInfo_t, [559](#)
 - gk_ReqInfo_t, [559](#)
 - gk_site_Close, [559](#)
 - gk_site_CreateComplete, [560](#)
 - gk_site_CreateStats, [561](#)
 - gk_site_GetMonitorTypes, [562](#)
 - gk_site_Init, [563](#)
 - gk_site_Open, [563](#)
 - gk_site_OpenStats, [564](#)
 - gk_site_Stage, [565](#)
 - gk_site_StageComplete, [565](#)
 - gk_site_StageStats, [566](#)
- ge64m
 - 64-bit Math Library, [500](#)
- gk_EntryInfo, [558](#)
- gk_EntryInfo_t
 - Gatekeeper Site Library, [559](#)
- gk_ReqInfo, [558](#)
- gk_ReqInfo_t
 - Gatekeeper Site Library, [559](#)
- gk_RTM.h, [682](#)
- gk_site_Close
 - Gatekeeper Site Library, [559](#)
- gk_site_CreateComplete
 - Gatekeeper Site Library, [560](#)
- gk_site_CreateStats
 - Gatekeeper Site Library, [561](#)
- gk_site_GetMonitorTypes
 - Gatekeeper Site Library, [562](#)
- gk_site_Init
 - Gatekeeper Site Library, [563](#)
- gk_site_Open
 - Gatekeeper Site Library, [563](#)
- gk_site_OpenStats
 - Gatekeeper Site Library, [564](#)
- gk_site_Stage
 - Gatekeeper Site Library, [565](#)
- gk_site_StageComplete
 - Gatekeeper Site Library, [565](#)
- gk_site_StageStats
 - Gatekeeper Site Library, [566](#)
- gk_SiteProtos.h, [682](#)
- gt64m
 - 64-bit Math Library, [501](#)
- hacl.h, [682](#)
- hacl_acl_entry, [548](#)
- hacl_acl_entry_t
 - HPSS Access Control List APIs, [549](#)
- hacl_ConvertACLToHACL
 - HPSS Access Control List APIs, [549](#)
- hacl_ConvertHACLPermsToPerms
 - HPSS Access Control List APIs, [550](#)
- hacl_ConvertHACLToACL
 - HPSS Access Control List APIs, [550](#)
- hacl_ConvertHACLToString
 - HPSS Access Control List APIs, [551](#)
- hacl_ConvertHACLTypeToType
 - HPSS Access Control List APIs, [552](#)
- hacl_ConvertPermsToHACLPerms
 - HPSS Access Control List APIs, [552](#)
- hacl_ConvertStringsToHACL
 - HPSS Access Control List APIs, [553](#)
- hacl_ConvertTypeToHACLType
 - HPSS Access Control List APIs, [554](#)
- hacl_DeleteHACL
 - HPSS Access Control List APIs, [554](#)
- hacl_GetHACL
 - HPSS Access Control List APIs, [555](#)

- HACL_MAX_ENTRY_SIZE
 - HPSS Access Control List APIs, [548](#)
- hacl_SetHACL
 - HPSS Access Control List APIs, [555](#)
- hacl_SortHACL
 - HPSS Access Control List APIs, [556](#)
- hacl_UpdateHACL
 - HPSS Access Control List APIs, [556](#)
- Hash Interface, [527](#)
 - HPSS_CKSUM_AFALG, [529](#)
 - HPSS_CKSUM_SSE42, [529](#)
 - HPSS_CKSUM_VPMSUM, [529](#)
 - hpss_HashAppend, [529](#)
 - hpss_HashAppendBuf, [530](#)
 - hpss_HashAppendStr, [530](#)
 - hpss_HashBaselNit, [531](#)
 - hpss_HashCreate, [531](#)
 - hpss_HashDecode, [532](#)
 - hpss_HashDelete, [532](#)
 - hpss_HashDigestLength, [533](#)
 - hpss_HashDigestToHex, [533](#)
 - hpss_HashDigestToString, [534](#)
 - hpss_HashDuplicate, [534](#)
 - hpss_HashEncode, [534](#)
 - hpss_HashExtract, [535](#)
 - hpss_HashFinish, [536](#)
 - hpss_HashFinishDigest, [536](#)
 - hpss_HashFinishHex, [537](#)
 - hpss_HashGetHWFlags, [537](#)
 - hpss_HashLoad, [538](#)
 - hpss_HashReset, [538](#)
 - hpss_HashType, [538](#)
 - hpss_HexStringToHashDigest, [539](#)
- high32m
 - 64-bit Math Library, [501](#)
- HPSS Access Control List APIs, [547](#)
 - hacl_acl_entry_t, [549](#)
 - hacl_ConvertACLToHACL, [549](#)
 - hacl_ConvertHACLPermsToPerms, [550](#)
 - hacl_ConvertHACLToACL, [550](#)
 - hacl_ConvertHACLToString, [551](#)
 - hacl_ConvertHACLTypeToType, [552](#)
 - hacl_ConvertPermsToHACLPerms, [552](#)
 - hacl_ConvertStringsToHACL, [553](#)
 - hacl_ConvertTypeToHACLType, [554](#)
 - hacl_DeleteHACL, [554](#)
 - hacl_GetHACL, [555](#)
 - HACL_MAX_ENTRY_SIZE, [548](#)
 - hacl_SetHACL, [555](#)
 - hacl_SortHACL, [556](#)
 - hacl_UpdateHACL, [556](#)
 - HPSS_EHACL_UNAUTH, [549](#)
- HPSS Client API Extensions, [505](#)
 - hpsscfx_ConfGetClientInterfaces, [507](#)
 - hpsscfx_ConfParse, [508](#)
 - hpsscfx_ConfSetDirPaths, [509](#)
 - hpsscfx_ConfSetFileName, [509](#)
 - hpsscfx_LookupHostName, [510](#)
 - hpsscfx_NetoptFindEntry, [511](#)
 - hpsscfx_NetoptGetWriteSize, [511](#)
 - hpsscfx_NetoptSetSock, [512](#)
 - hpsscfx_PatternMatch, [512](#)
 - MIN_RESTRICTED_PORT, [506](#)
- HPSS Config File API, [539](#)
 - CFG_COMPOUND_TYPE, [544](#)
 - CFG_FLAG_TYPE, [544](#)
 - CFG_SIMPLE_TYPE, [544](#)
 - HPSS_CFG_MAX_CHARS_PER_LINE, [543](#)
 - HPSS_CFG_MAX_LEVEL, [543](#)
 - hpss_cfg_stanza_t, [543](#)
 - HPSS_CFG_STANZA_TYPE, [543](#)
 - hpss_CfgFindKey, [544](#)
 - hpss_CfgFindToken, [544](#)
 - hpss_CfgFree, [545](#)
 - hpss_CfgParse, [545](#)
 - hpsscfx_ConfParse, [546](#)
- HPSS Config File Template API, [513](#)
- HPSS Environment Variable Module, [473](#)
 - hpss_Envcpy, [474](#)
 - hpss_EnvLocalUpdate, [474](#)
 - hpss_EnvReInitialize, [475](#)
 - hpss_Getenv, [475](#)
- HPSS Networking Library, [475](#)
 - hpss_net_accept, [476](#)
 - hpss_net_addrmatch, [477](#)
 - hpss_net_bind, [477](#)
 - hpss_net_connect, [479](#)
 - hpss_net_generatemask, [480](#)
 - hpss_net_get_family_option, [480](#)
 - hpss_net_get_family_string, [481](#)
 - hpss_net_getinaddr, [481](#)
 - hpss_net_getnameinfo, [481](#)
 - hpss_net_getpeername, [482](#)
 - hpss_net_getport, [483](#)
 - hpss_net_getuniversaladdress, [483](#)
 - hpss_net_initaddr, [484](#)
 - hpss_net_listen, [484](#)
 - hpss_net_setport, [485](#)
 - hpss_net_socket, [485](#)
 - hpss_net_strerror, [486](#)
 - hpss_net_universaladdressstoport, [486](#)
- HPSS Object, [394](#)
 - hpss_GetObjFromHandle, [394](#)
 - hpss_GetObjType, [395](#)
 - hpss_GetPathHandle, [395](#)
 - hpss_GetPathHandleObjHash, [396](#)
 - hpss_GetPathObjHash, [396](#)
- HPSS Statistics, [389](#)
 - hpss_GetSubSysLimits, [390](#)
 - hpss_GetSubSysStats, [391](#)
 - hpss_ResetSubSysStats, [391](#)
 - hpss_Statfs, [392](#)
 - hpss_Statfs64, [392](#)
 - hpss_Statvfs, [393](#)
 - hpss_Statvfs64, [393](#)
- HPSS XML Interface, [487](#)

- hpss_ChompXMLHeader, [488](#)
- hpss_CreateXMLWhere, [488](#)
- hpss_CreateXQueryDelete, [489](#)
- hpss_CreateXQueryGet, [489](#)
- hpss_CreateXQueryUpdate, [490](#)
- hpss_ValidXPath, [491](#)
- hpss_XMLtoAttr, [492](#)
- hpss_XPathBranch, [492](#)
- hpss_XPathLeaf, [493](#)
- hpss_XPathRBranch, [494](#)
- hpss_XPathRoot, [494](#)
- hpss_XPathtoXML, [495](#)
- hpss_AbortIOByMoverId
 - Client API, [107](#)
- hpss_AbortIOByMoverName
 - Client API, [107](#)
- hpss_AbortIOByRequestId
 - Client API, [108](#)
- hpss_AbortIOByRequestMatch
 - Client API, [108](#)
- hpss_AbortIOByUserld
 - Client API, [110](#)
- hpss_Access
 - Client API, [111](#)
 - File Attribute, [324](#)
- hpss_AccessHandle
 - Client API, [111](#)
 - File Attribute, [324](#)
- hpss_AcctCodeToName
 - Client API, [112](#)
- hpss_AcctNameToCode
 - Client API, [113](#)
- hpss_AdminStateString
 - hpss_String.c, [731](#)
- hpss_AllocStateString
 - hpss_String.c, [731](#)
- hpss_api.h, [684](#)
- hpss_AuthenticatorTypeFromString
 - hpss_String.c, [732](#)
- hpss_AuthenticatorTypeString
 - hpss_String.c, [732](#)
- hpss_AuthnMechTypeFromString
 - hpss_String.c, [733](#)
- hpss_AuthnMechTypeString
 - hpss_String.c, [733](#)
- hpss_AuthzMechTypeFromString
 - hpss_String.c, [733](#)
- hpss_AuthzMechTypeString
 - hpss_String.c, [734](#)
- hpss_BuildLevelString
 - Client API, [113](#)
- hpss_Bytecount32
 - hpss_String.c, [734](#)
- hpss_Bytecount64
 - hpss_String.c, [735](#)
- HPSS_CFG_MAX_CHARS_PER_LINE
 - HPSS Config File API, [543](#)
- HPSS_CFG_MAX_LEVEL
 - HPSS Config File API, [543](#)
- hpss_cfg_stanza, [542](#)
- hpss_cfg_stanza_t
 - HPSS Config File API, [543](#)
- HPSS_CFG_STANZA_TYPE
 - HPSS Config File API, [543](#)
- hpss_CfgFindKey
 - HPSS Config File API, [544](#)
- hpss_CfgFindToken
 - HPSS Config File API, [544](#)
- hpss_CfgFree
 - HPSS Config File API, [545](#)
- hpss_CfgParse
 - HPSS Config File API, [545](#)
- hpss_Chacct
 - Client API, [114](#)
 - File Attribute, [325](#)
- hpss_ChacctByName
 - Client API, [115](#)
 - File Attribute, [326](#)
- hpss_Chdir
 - Client API, [115](#)
 - Working Directory, [376](#)
- hpss_Chmod
 - Client API, [117](#)
 - File Attribute, [326](#)
- hpss_ChompXMLHeader
 - HPSS XML Interface, [488](#)
- hpss_Chown
 - Client API, [117](#)
 - File Attribute, [327](#)
- hpss_Chroot
 - Client API, [118](#)
 - Working Directory, [377](#)
- HPSS_CKSUM_AFALG
 - Hash Interface, [529](#)
- HPSS_CKSUM_SSE42
 - Hash Interface, [529](#)
- HPSS_CKSUM_VPMSUM
 - Hash Interface, [529](#)
- hpss_ClearLastHPSSErrno
 - Client API, [119](#)
- hpss_ClientAPIInit
 - Client API, [119](#)
 - Client API Control, [380](#)
- hpss_ClientAPIReset
 - Client API, [119](#)
 - Client API Control, [382](#)
- hpss_Close
 - Client API, [120](#)
 - File Open, Create, and Close, [285](#)
- hpss_Closedir
 - Client API, [120](#)
 - Directory Data, [366](#)
- hpss_ConvertIdsToNames
 - Client API, [121](#)
- hpss_ConvertNamesTolds
 - Client API, [122](#)

- hpss_CopyFile
 - Client API, [123](#)
 - File Data, [300](#)
- hpss_CopyrightStrings
 - Client API, [123](#)
- hpss_Creat
 - Client API, [123](#)
 - File Open, Create, and Close, [285](#)
- hpss_Create
 - Client API, [124](#)
 - File Open, Create, and Close, [286](#)
- hpss_CreateHandle
 - Client API, [125](#)
 - File Open, Create, and Close, [287](#)
- hpss_CreateWithHints
 - POSIX APIs, [519](#)
- hpss_CreateXMLWhere
 - HPSS XML Interface, [488](#)
- hpss_CreateXQueryDelete
 - HPSS XML Interface, [489](#)
- hpss_CreateXQueryGet
 - HPSS XML Interface, [489](#)
- hpss_CreateXQueryUpdate
 - HPSS XML Interface, [490](#)
- hpss_DeleteACL
 - ACLs, [354](#)
 - Client API, [126](#)
- hpss_DeleteACLHandle
 - ACLs, [355](#)
 - Client API, [127](#)
- hpss_dirent, [701](#)
- hpss_dirent.h, [700](#)
- hpss_DriveFlagString
 - hpss_String.c, [735](#)
- hpss_DriveStateString
 - hpss_String.c, [735](#)
- hpss_DriveTypeString
 - hpss_String.c, [736](#)
- HPSS_EHACL_UNAUTH
 - HPSS Access Control List APIs, [549](#)
- hpss_Envcpy
 - HPSS Environment Variable Module, [474](#)
- hpss_EnvLocalUpdate
 - HPSS Environment Variable Module, [474](#)
- hpss_EnvReInitialize
 - HPSS Environment Variable Module, [475](#)
- hpss_errno_state, [80](#)
- hpss_errno_state_t
 - Client API, [90](#)
- hpss_ErrnoName
 - hpss_String.c, [736](#)
- hpss_Fclear
 - Client API, [128](#)
 - File Data, [300](#)
- hpss_FclearOffset
 - Client API, [128](#)
 - File Data, [301](#)
- hpss_Fclose
 - Buffered Data, [398](#)
 - Client API, [130](#)
- hpss_Fcntl
 - Buffered Data, [398](#)
 - Client API, [131](#)
- hpss_FdelFileDigestList
 - Client API, [132](#)
- hpss_Fflush
 - Buffered Data, [399](#)
 - Client API, [132](#)
- hpss_Fgetc
 - Buffered Data, [400](#)
 - Client API, [133](#)
- hpss_FgetFileDigest
 - Client API, [134](#)
- hpss_FgetFileDigestList
 - Client API, [134](#)
- hpss_Fgets
 - Buffered Data, [401](#)
 - Client API, [135](#)
- HPSS_FILE
 - Client API, [90](#)
- hpss_file_hash_digest, [81](#)
- HPSS_FILE_HASH_DIGEST_ACTIVE
 - Client API, [92](#)
- hpss_file_hash_digest_list, [81](#)
- HPSS_FILE_HASH_DIGEST_VALID
 - Client API, [92](#)
- hpss_file_hash_flags
 - Client API, [91](#)
- hpss_file_hash_flags_t
 - Client API, [91](#)
- HPSS_FILE_HASH_GENERATED
 - Client API, [92](#)
- HPSS_FILE_HASH_MIGRATE_VERIFIED
 - Client API, [92](#)
- HPSS_FILE_HASH_SET_BY_USER
 - Client API, [92](#)
- HPSS_FILE_HASH_SKIP_VERIFICATION
 - Client API, [92](#)
- hpss_file_hash_stripe_flags
 - Client API, [92](#)
- hpss_FileGetAttributes
 - Client API, [136](#)
 - File Attribute, [327](#)
- hpss_FileGetAttributesBitfile
 - Client API, [137](#)
 - File Attribute, [328](#)
- hpss_FileGetAttributesHandle
 - Client API, [138](#)
 - File Attribute, [329](#)
- hpss_FileGetDigestHandle
 - Client API, [139](#)
- hpss_FileGetDigestListHandle
 - Client API, [139](#)
- hpss_FileGetXAttributes
 - Client API, [140](#)
 - File Attribute, [330](#)

- hpss_FileGetXAttributesHandle
 - Client API, [142](#)
 - File Attribute, [331](#)
- hpss_FileSetAttributes
 - Client API, [143](#)
 - File Attribute, [332](#)
- hpss_FileSetAttributesBitfile
 - Client API, [145](#)
 - File Attribute, [334](#)
- hpss_FileSetAttributesHandle
 - Client API, [146](#)
 - File Attribute, [335](#)
- hpss_FileSetCOS
 - Client API, [147](#)
 - File Attribute, [336](#)
- hpss_FileSetCOSHandle
 - Client API, [148](#)
 - File Attribute, [337](#)
- hpss_FilesetCreate
 - Client API, [149](#)
 - Fileset and Junction, [314](#)
- hpss_FilesetDelete
 - Client API, [150](#)
 - Fileset and Junction, [315](#)
- hpss_FilesetGetAttributes
 - Client API, [150](#)
 - Fileset and Junction, [316](#)
- hpss_FilesetListAll
 - Client API, [151](#)
 - Fileset and Junction, [317](#)
- hpss_FilesetSetAttributes
 - Client API, [152](#)
 - Fileset and Junction, [317](#)
- hpss_Fopen
 - Buffered Data, [402](#)
 - Client API, [152](#)
- hpss_Fpreallocate
 - Client API, [154](#)
 - File Data, [301](#)
- hpss_Fread
 - Buffered Data, [403](#)
 - Client API, [154](#)
- hpss_Fseek
 - Buffered Data, [404](#)
 - Client API, [155](#)
- hpss_FsetFileDigest
 - Client API, [156](#)
- hpss_FsetFileDigestList
 - Client API, [157](#)
- hpss_Fstat
 - Client API, [158](#)
 - File Attribute, [338](#)
- hpss_Fsync
 - Buffered Data, [405](#)
 - Client API, [159](#)
- hpss_Ftell
 - Buffered Data, [405](#)
 - Client API, [159](#)
- hpss_Ftruncate
 - Client API, [159](#)
 - File Data, [302](#)
- hpss_Fwrite
 - Buffered Data, [406](#)
 - Client API, [160](#)
- hpss_GetAcct
 - Client API, [161](#)
- hpss_GetAcctName
 - Client API, [162](#)
- hpss_GetACL
 - ACLs, [356](#)
 - Client API, [162](#)
- hpss_GetACLHandle
 - ACLs, [356](#)
 - Client API, [163](#)
- hpss_GetAllSubsystems
 - Client API, [164](#)
- hpss_GetAsyncStatus
 - Client API, [164](#)
- hpss_GetAttrHandle
 - Client API, [165](#)
 - File Attribute, [338](#)
- hpss_GetAuthType
 - hpss_Mech.c, [716](#)
- hpss_GetBatchAsynchStatus
 - Client API, [166](#)
- hpss_GetCallbackStatus
 - Client API, [166](#)
- hpss_GetConfiguration
 - Client API, [167](#)
 - Client API Control, [382](#)
- hpss_Getcwd
 - Client API, [167](#)
 - Working Directory, [378](#)
- hpss_Getdents
 - Client API, [168](#)
 - Directory Data, [367](#)
- hpss_GetdentsAttrs
 - Client API, [168](#)
 - Directory Data, [368](#)
- hpss_GetDistFile
 - Client API, [169](#)
- hpss_Getenv
 - HPSS Environment Variable Module, [475](#)
- hpss_Getenv.c, [701](#)
 - hpss_GetenvDefault, [702](#)
- hpss_Getenv.h, [702](#)
 - hpss_GetenvDefault, [703](#)
- hpss_GetenvDefault
 - hpss_Getenv.c, [702](#)
 - hpss_Getenv.h, [703](#)
- hpss_GetFileNotrunc
 - Client API, [170](#)
 - File Attribute, [339](#)
- hpss_GetFullPath
 - Client API, [170](#)
- hpss_GetFullPathBitfile

- Client API, [171](#)
- hpss_GetFullPathBitfileLen
 - Client API, [171](#)
- hpss_GetFullPathHandle
 - Client API, [172](#)
- hpss_GetFullPathHandleLen
 - Client API, [173](#)
- hpss_GetFullPathLen
 - Client API, [173](#)
- hpss_GetGID
 - Client API, [174](#)
- hpss_GetIntersectedList
 - Client API, [174](#)
- hpss_GetJunctionAttrs
 - Client API, [175](#)
 - Fileset and Junction, [318](#)
- hpss_GetJunctions
 - Client API, [175](#)
 - Fileset and Junction, [319](#)
- hpss_GetLastHPSSErrno
 - Client API, [176](#)
- hpss_getline
 - hpss_String.c, [736](#)
- hpss_GetListAttrs
 - Client API, [176](#)
 - File Attribute, [339](#)
- hpss_GetNextIORquestID
 - Client API, [177](#)
- hpss_GetObjFromHandle
 - Client API, [177](#)
 - HPSS Object, [394](#)
- hpss_GetObjType
 - Client API, [178](#)
 - HPSS Object, [395](#)
- hpss_GetPathHandle
 - Client API, [178](#)
 - HPSS Object, [395](#)
- hpss_GetPathHandleObjHash
 - Client API, [179](#)
 - HPSS Object, [396](#)
- hpss_GetPathObjHash
 - Client API, [179](#)
 - HPSS Object, [396](#)
- hpss_GetRawAttrHandle
 - Client API, [180](#)
 - File Attribute, [340](#)
- hpss_GetSubSysLimits
 - Client API, [180](#)
 - HPSS Statistics, [390](#)
- hpss_GetSubSysStats
 - Client API, [181](#)
 - HPSS Statistics, [391](#)
- hpss_GetThreadUcred
 - Client API, [181](#)
 - Working Directory, [378](#)
- hpss_GetTrashcan
 - Client API, [182](#)
 - Trashcan, [440](#)
- hpss_GetTrashcanHandle
 - Client API, [183](#)
 - Trashcan, [441](#)
- hpss_GetTrashSettings
 - Client API, [184](#)
 - Trashcan, [442](#)
- hpss_GetUID
 - Client API, [184](#)
- hpss_global_fsent_t, [81](#)
- hpss_hash.c, [703](#)
 - ADLER32_BASE, [705](#)
- hpss_hash.h, [705](#)
- HPSS_HASH_SINGLE
 - Client API, [92](#)
- HPSS_HASH_SINGLE_ONLY
 - Client API, [92](#)
- HPSS_HASH_STRIPE
 - Client API, [92](#)
- HPSS_HASH_STRIPE_ONLY
 - Client API, [92](#)
- hpss_HashAppend
 - Hash Interface, [529](#)
- hpss_HashAppendBuf
 - Hash Interface, [530](#)
- hpss_HashAppendStr
 - Hash Interface, [530](#)
- hpss_HashBaselnit
 - Hash Interface, [531](#)
- hpss_HashCreate
 - Hash Interface, [531](#)
- hpss_HashDecode
 - Hash Interface, [532](#)
- hpss_HashDelete
 - Hash Interface, [532](#)
- hpss_HashDigestLength
 - Hash Interface, [533](#)
- hpss_HashDigestToHex
 - Hash Interface, [533](#)
- hpss_HashDigestToString
 - Hash Interface, [534](#)
- hpss_HashDuplicate
 - Hash Interface, [534](#)
- hpss_HashEncode
 - Hash Interface, [534](#)
- hpss_HashExtract
 - Hash Interface, [535](#)
- hpss_HashFinish
 - Hash Interface, [536](#)
- hpss_HashFinishDigest
 - Hash Interface, [536](#)
- hpss_HashFinishHex
 - Hash Interface, [537](#)
- hpss_HashFlagsToString
 - Client API, [184](#)
- hpss_HashGetHWFlags
 - Hash Interface, [537](#)
- hpss_HashLoad
 - Hash Interface, [538](#)

- hpss_HashReset
 - Hash Interface, [538](#)
- hpss_HashType
 - Hash Interface, [538](#)
- hpss_HashTypeString
 - hpss_String.c, [737](#)
- hpss_HexString
 - hpss_String.c, [737](#)
- hpss_HexStringToHashDigest
 - Hash Interface, [539](#)
- hpss_htable.h, [706](#)
 - hpss_htable_node_t, [707](#)
- hpss_htable_constants
 - Data Containers, [453](#)
- hpss_htable_node_t
 - hpss_htable.h, [707](#)
- hpss_htable.h, [707](#)
 - hpss_htable_foreach_retval, [711](#)
 - hpss_htable_foreach_retval_e, [710](#)
- hpss_htable_alloc_func_t
 - Data Containers, [451](#)
- hpss_htable_append
 - Data Containers, [454](#)
- hpss_htable_clear
 - Data Containers, [454](#)
- hpss_htable_custom_init
 - Data Containers, [455](#)
- hpss_htable_destructor_t
 - Data Containers, [451](#)
- hpss_htable_empty
 - Data Containers, [455](#)
- hpss_htable_foreach
 - Data Containers, [456](#)
- hpss_htable_foreach_range
 - Data Containers, [456](#)
- hpss_htable_foreach_retval
 - hpss_htable.h, [711](#)
- hpss_htable_foreach_retval_e
 - hpss_htable.h, [710](#)
- hpss_htable_get_current
 - Data Containers, [457](#)
- hpss_htable_get_current_r
 - Data Containers, [457](#)
- hpss_htable_get_node
 - Data Containers, [458](#)
- hpss_htable_get_size
 - Data Containers, [458](#)
- hpss_htable_halt_iterator
 - Data Containers, [459](#)
- hpss_htable_init
 - Data Containers, [459](#)
- hpss_htable_init_iterator
 - Data Containers, [459](#)
- hpss_htable_init_iterator_r
 - Data Containers, [460](#)
- hpss_htable_init_iterator_range
 - Data Containers, [460](#)
- hpss_htable_init_iterator_range_r
 - Data Containers, [461](#)
- hpss_htable_insert
 - Data Containers, [461](#)
- hpss_htable_iterator_token_h
 - Data Containers, [451](#)
- hpss_htable_join
 - Data Containers, [462](#)
- hpss_htable_mapped_func_t
 - Data Containers, [451](#)
- hpss_htable_prepend
 - Data Containers, [462](#)
- hpss_htable_reinit_iterator
 - Data Containers, [463](#)
- hpss_htable_reinit_iterator_range
 - Data Containers, [463](#)
- hpss_htable_remove
 - Data Containers, [464](#)
- hpss_htable_remove_safe
 - Data Containers, [464](#)
- hpss_htable_t
 - Data Containers, [452](#)
- hpss_InsertDistFile
 - Client API, [185](#)
- hpss_irbtree, [449](#)
- hpss_irbtree.c, [711](#)
- hpss_irbtree.h, [713](#)
- hpss_irbtree_clear
 - Data Containers, [465](#)
- hpss_irbtree_color
 - Data Containers, [454](#)
- hpss_irbtree_container
 - Data Containers, [450](#)
- hpss_irbtree_empty
 - Data Containers, [465](#)
- hpss_irbtree_find
 - Data Containers, [465](#)
- hpss_irbtree_find_cmp
 - Data Containers, [466](#)
- hpss_irbtree_foreach
 - Data Containers, [467](#)
- hpss_irbtree_foreach_node
 - Data Containers, [467](#)
- hpss_irbtree_foreach_node_prev
 - Data Containers, [468](#)
- hpss_irbtree_init
 - Data Containers, [468](#)
- hpss_irbtree_insert
 - Data Containers, [469](#)
- hpss_irbtree_insert_multi
 - Data Containers, [469](#)
- hpss_irbtree_lower_bound
 - Data Containers, [469](#)
- hpss_irbtree_max
 - Data Containers, [470](#)
- hpss_irbtree_min
 - Data Containers, [470](#)
- hpss_irbtree_node, [450](#)
- hpss_irbtree_node_callback_t

- Data Containers, [452](#)
- hpss_irbtree_node_comparator_t
 - Data Containers, [452](#)
- hpss_irbtree_node_destructor_t
 - Data Containers, [453](#)
- hpss_irbtree_node_next
 - Data Containers, [471](#)
- hpss_irbtree_node_prev
 - Data Containers, [471](#)
- hpss_irbtree_node_printer_t
 - Data Containers, [453](#)
- hpss_irbtree_print
 - Data Containers, [471](#)
- hpss_irbtree_remove
 - Data Containers, [472](#)
- hpss_irbtree_size
 - Data Containers, [472](#)
- hpss_irbtree_upper_bound
 - Data Containers, [473](#)
- hpss_junction_ent_t, [81](#)
- hpss_JunctionCreate
 - Client API, [185](#)
 - Fileset and Junction, [319](#)
- hpss_JunctionCreateHandle
 - Client API, [186](#)
 - Fileset and Junction, [320](#)
- hpss_JunctionDelete
 - Client API, [187](#)
 - Fileset and Junction, [321](#)
- hpss_JunctionDeleteHandle
 - Client API, [187](#)
 - Fileset and Junction, [321](#)
- hpss_LabelTypeString
 - hpss_String.c, [738](#)
- hpss_Link
 - Client API, [188](#)
 - File Name, [347](#)
- hpss_LinkHandle
 - Client API, [189](#)
 - File Name, [347](#)
- hpss_LinkHandleParent
 - Client API, [189](#)
 - File Name, [349](#)
- hpss_LoadDefaultThreadState
 - Client API, [190](#)
 - Client API Control, [382](#)
- hpss_LoadThreadState
 - Client API, [191](#)
- hpss_LocateRootCS
 - Client API, [192](#)
 - Client API Location Services, [524](#)
- hpss_LocateServerByID
 - Client API, [192](#)
 - Client API Location Services, [525](#)
- hpss_LocateServersByType
 - Client API Location Services, [525](#)
- hpss_LocateSiteById
 - Client API, [193](#)
- Client API Location Services, [526](#)
- hpss_LocateSiteByName
 - Client API, [193](#)
 - Client API Location Services, [527](#)
- hpss_LookupRootCS
 - Client API, [194](#)
- hpss_Lseek
 - Client API, [194](#)
 - File Data, [303](#)
- hpss_Lstat
 - Client API, [195](#)
 - File Attribute, [341](#)
- hpss_Mech.c, [715](#)
 - hpss_GetAuthType, [716](#)
- hpss_MediaTypeString
 - hpss_String.c, [738](#)
- hpss_Migrate
 - Client API, [196](#)
- HPSS_MIGRATE_FORCE
 - Client API, [87](#)
- HPSS_MIGRATE_HPSS_ONLY
 - Client API, [87](#)
- HPSS_MIGRATE_NO_COPY
 - Client API, [87](#)
- HPSS_MIGRATE_PURGE_DATA
 - Client API, [87](#)
- hpss_MiscAdmin
 - Client API, [196](#)
 - Client API Control, [383](#)
- hpss_Mkdir
 - Client API, [197](#)
 - Directory Creation and Deletion, [361](#)
- hpss_MkdirHandle
 - Client API, [198](#)
 - Directory Creation and Deletion, [362](#)
- hpss_MountModeString
 - hpss_String.c, [738](#)
- hpss_net.c, [716](#)
 - __hpss_net_maskaddr32, [718](#)
 - __hpss_net_setscopeid, [718](#)
 - hpss_net_getaddrinfo, [719](#)
 - hpss_net_getaddrinfo_ex, [720](#)
 - hpss_net_getsockname, [721](#)
- hpss_net.h, [721](#)
 - __hpss_net_maskaddr32, [723](#)
 - __hpss_net_setscopeid, [723](#)
 - hpss_net_getaddrinfo, [724](#)
 - hpss_net_getaddrinfo_ex, [724](#)
 - hpss_net_getsockname, [725](#)
- hpss_net_accept
 - HPSS Networking Library, [476](#)
- hpss_net_addrmatch
 - HPSS Networking Library, [477](#)
- hpss_net_bind
 - HPSS Networking Library, [477](#)
- hpss_net_connect
 - HPSS Networking Library, [479](#)
- hpss_net_generatemask

- HPSS Networking Library, [480](#)
- hpss_net_get_family_option
 - HPSS Networking Library, [480](#)
- hpss_net_get_family_string
 - HPSS Networking Library, [481](#)
- hpss_net_getaddrinfo
 - hpss_net.c, [719](#)
 - hpss_net.h, [724](#)
- hpss_net_getaddrinfo_ex
 - hpss_net.c, [720](#)
 - hpss_net.h, [724](#)
- hpss_net_getinaddr
 - HPSS Networking Library, [481](#)
- hpss_net_getnameinfo
 - HPSS Networking Library, [481](#)
- hpss_net_getpeername
 - HPSS Networking Library, [482](#)
- hpss_net_getport
 - HPSS Networking Library, [483](#)
- hpss_net_getsockname
 - hpss_net.c, [721](#)
 - hpss_net.h, [725](#)
- hpss_net_getuniversaladdress
 - HPSS Networking Library, [483](#)
- hpss_net_initaddr
 - HPSS Networking Library, [484](#)
- hpss_net_listen
 - HPSS Networking Library, [484](#)
- hpss_net_setport
 - HPSS Networking Library, [485](#)
- hpss_net_socket
 - HPSS Networking Library, [485](#)
- hpss_net_strerror
 - HPSS Networking Library, [486](#)
- hpss_net_universaladdressstoport
 - HPSS Networking Library, [486](#)
- HPSS_O_EXCL
 - Client API, [88](#)
- HPSS_O_NO_TAPE
 - Client API, [88](#)
- HPSS_O_STAGE_ASYNC
 - Client API, [88](#)
- HPSS_O_STAGE_BKGRD
 - Client API, [88](#)
- HPSS_O_STAGE_NONE
 - Client API, [88](#)
- hpss_Open
 - Client API, [199](#)
 - File Open, Create, and Close, [288](#)
- hpss_OpenBitfile
 - Client API, [200](#)
 - File Open, Create, and Close, [289](#)
- hpss_OpenBitfileVAttr
 - Client API, [202](#)
 - File Open, Create, and Close, [290](#)
- hpss_Opendir
 - Client API, [203](#)
 - Directory Data, [369](#)
- hpss_OpendirHandle
 - Client API, [204](#)
 - Directory Data, [369](#)
- hpss_OpenHandle
 - Client API, [205](#)
 - File Open, Create, and Close, [291](#)
- hpss_OpenWithHints
 - POSIX APIs, [520](#)
- hpss_OpStateString
 - hpss_String.c, [739](#)
- hpss_PingCore
 - Client API, [206](#)
- hpss_pio_gapinfo_s, [82](#)
- HPSS_PIO_HANDLE_GAP
 - Client API, [88](#)
- HPSS_PIO_MVR_SELECT
 - Client API, [93](#)
- HPSS_PIO_NONE
 - Client API, [92](#)
- hpss_pio_operation_t
 - Client API, [92](#)
- hpss_pio_options_t
 - Client API, [91](#)
- hpss_pio_params_s, [82](#)
- HPSS_PIO_PORT_RANGE
 - Client API, [88](#)
- HPSS_PIO_PUSH
 - Client API, [89](#)
- HPSS_PIO_READ
 - Client API, [92](#)
- HPSS_PIO_TCPIP
 - Client API, [93](#)
- hpss_pio_transport_t
 - Client API, [92](#)
- HPSS_PIO_USE_API_HOSTNAME
 - Client API, [89](#)
- HPSS_PIO_WRITE
 - Client API, [92](#)
- hpss_PIOEnd
 - Client API, [207](#)
 - Parallel I/O, [426](#)
- hpss_PIOExecute
 - Client API, [207](#)
 - Parallel I/O, [427](#)
- hpss_PIOExecuteItem
 - Client API, [208](#)
 - Parallel I/O, [427](#)
- hpss_PIOExportGrp
 - Client API, [209](#)
 - Parallel I/O, [428](#)
- hpss_PIOFinalize
 - Client API, [210](#)
 - Parallel I/O, [429](#)
- hpss_PIOGetPartError
 - Client API, [210](#)
 - Parallel I/O, [429](#)
- hpss_PIOGetRequestID
 - Client API, [211](#)

- Parallel I/O, [430](#)
- hpss_PIOImportGrp
 - Client API, [211](#)
 - Parallel I/O, [430](#)
- hpss_PIOInitCtx
 - Client API, [212](#)
 - Simplified Parallel I/O, [433](#)
- hpss_PIOReadFile
 - Client API, [212](#)
 - Simplified Parallel I/O, [434](#)
- hpss_PIOReadFileMulti
 - Client API, [213](#)
 - Simplified Parallel I/O, [434](#)
- hpss_PIORegister
 - Client API, [214](#)
 - Parallel I/O, [431](#)
- hpss_PIOResetCtx
 - Client API, [214](#)
 - Simplified Parallel I/O, [435](#)
- hpss_PIOSetBlockSize
 - Client API, [215](#)
 - Simplified Parallel I/O, [435](#)
- hpss_PIOSetDefaultCtx
 - Client API, [215](#)
 - Simplified Parallel I/O, [436](#)
- hpss_PIOSetHandleGaps
 - Client API, [216](#)
 - Simplified Parallel I/O, [436](#)
- hpss_PIOSetOffset
 - Simplified Parallel I/O, [437](#)
- hpss_PIOSetStripeWidth
 - Client API, [216](#)
 - Simplified Parallel I/O, [437](#)
- hpss_PIOSetUseAPIHost
 - Client API, [217](#)
 - Simplified Parallel I/O, [438](#)
- hpss_PIOStart
 - Client API, [217](#)
 - Parallel I/O, [432](#)
- hpss_PIOWriteFile
 - Client API, [218](#)
 - Simplified Parallel I/O, [438](#)
- hpss_PIOWriteFileMulti
 - Client API, [218](#)
 - Simplified Parallel I/O, [439](#)
- hpss_posix_api.h, [726](#)
- hpss_PosixCreate
 - POSIX APIs, [521](#)
- hpss_PosixFstat
 - POSIX APIs, [521](#)
- hpss_PosixLseek
 - POSIX APIs, [522](#)
- hpss_PosixLstat
 - POSIX APIs, [522](#)
- hpss_PosixOpen
 - POSIX APIs, [523](#)
- hpss_PosixStat
 - POSIX APIs, [523](#)
- hpss_PREFERRED_READ_ORDER
 - Client API, [219](#)
 - File Data, [303](#)
- hpss_ProtLevelFromString
 - hpss_String.c, [739](#)
- hpss_ProtLevelString
 - hpss_String.c, [739](#)
- hpss_Purge
 - Client API, [220](#)
- hpss_PurgeLock
 - Client API, [220](#)
 - File Attribute, [341](#)
- hpss_PurgeLoginCred
 - Authentication, [387](#)
 - Client API, [221](#)
- hpss_PurgeOnMigrate
 - Client API, [222](#)
 - File Attribute, [342](#)
- hpss_PVLActivityStateString
 - hpss_String.c, [740](#)
- hpss_PVLJobStateString
 - hpss_String.c, [740](#)
- hpss_PVRTypeString
 - hpss_String.c, [741](#)
- hpss_Read
 - Client API, [222](#)
 - File Data, [305](#)
- hpss_ReadAttrs
 - Client API, [223](#)
 - Directory Data, [370](#)
- hpss_ReadAttrsHandle
 - Client API, [224](#)
 - Directory Data, [370](#)
- hpss_ReadAttrsPlus
 - Client API, [224](#)
 - Directory Data, [371](#)
- hpss_Readdir
 - Client API, [225](#)
 - Directory Data, [372](#)
- hpss_readdir_flags_t
 - Client API, [93](#)
- HPSS_READDIR_GETATTRS
 - Client API, [93](#)
- HPSS_READDIR_NFS
 - Client API, [93](#)
- HPSS_READDIR_NONE
 - Client API, [93](#)
- HPSS_READDIR_RAW
 - Client API, [93](#)
- hpss_ReaddirHandle
 - Client API, [226](#)
 - Directory Data, [373](#)
- hpss_ReaddirPlus
 - Client API, [227](#)
 - Directory Data, [374](#)
- hpss_ReadItem
 - Client API, [228](#)
 - File Data, [306](#)

- hpss_Readlink
 - Client API, [228](#)
- hpss_ReadlinkHandle
 - Client API, [229](#)
- hpss_ReadList
 - Client API, [230](#)
- hpss_ReadQueueAdd
 - Client API, [230](#)
 - File Data, [306](#)
- hpss_ReadQueueAddMany
 - Client API, [231](#)
 - File Data, [307](#)
- hpss_ReadQueueCreateContext
 - Client API, [231](#)
 - File Data, [308](#)
- hpss_ReadQueueList
 - Client API, [232](#)
 - File Data, [308](#)
- hpss_ReadQueueRemove
 - Client API, [232](#)
 - File Data, [309](#)
- hpss_ReadQueueRemoveContext
 - Client API, [233](#)
 - File Data, [309](#)
- hpss_ReadRawAttrsHandle
 - Client API, [233](#)
 - Directory Data, [375](#)
- hpss_ReadTrash
 - Trashcan, [442](#)
- hpss_RemoveIntersectionFromList
 - Client API, [234](#)
- hpss_Rename
 - Client API, [235](#)
 - File Name, [350](#)
- hpss_RenameHandle
 - Client API, [236](#)
 - File Name, [351](#)
- hpss_ReopenBitfile
 - Client API, [237](#)
 - File Open, Create, and Close, [293](#)
- hpss_ResetSubSysStats
 - Client API, [238](#)
 - HPSS Statistics, [391](#)
- hpss_Rewinddir
 - Client API, [238](#)
 - Directory Data, [375](#)
- hpss_Rmdir
 - Client API, [239](#)
 - Directory Creation and Deletion, [362](#)
- hpss_RmdirHandle
 - Client API, [239](#)
 - Directory Creation and Deletion, [363](#)
- hpss_RmdirHandleImmediate
 - Client API, [240](#)
 - Directory Creation and Deletion, [364](#)
- hpss_RmdirImmediate
 - Client API, [241](#)
 - Directory Creation and Deletion, [365](#)
- hpss_Rumble
 - api_rddir.c, [662](#)
- hpss_ServerTypeString
 - hpss_String.c, [741](#)
- hpss_SetAcct
 - Client API, [242](#)
- hpss_SetAcctByName
 - Client API, [242](#)
- hpss_SetACL
 - ACLs, [357](#)
 - Client API, [243](#)
- hpss_SetACLHandle
 - ACLs, [358](#)
 - Client API, [243](#)
- hpss_SetAPILogLevel
 - Client API, [244](#)
 - Client API Control, [384](#)
- hpss_SetAPILogPath
 - Client API, [245](#)
 - Client API Control, [385](#)
- hpss_SetApplicationName
 - Client API, [245](#)
 - Client API Control, [385](#)
- hpss_SetAppLoginCred
 - Authentication, [387](#)
 - Client API, [246](#)
- hpss_SetAttrHandle
 - Client API, [246](#)
 - File Attribute, [343](#)
- hpss_SetAuditInfo
 - Authentication, [388](#)
 - Client API, [247](#)
- hpss_SetConfiguration
 - Client API, [248](#)
 - Client API Control, [386](#)
- hpss_SetCOSByHints
 - Client API, [248](#)
 - File Attribute, [344](#)
- hpss_SetFileNotrunc
 - Client API, [249](#)
 - File Attribute, [344](#)
- hpss_SetFileOffset
 - Client API, [250](#)
 - File Data, [310](#)
- hpss_SetGID
 - Client API, [250](#)
- hpss_SetLoginCred
 - Authentication, [389](#)
 - Client API, [251](#)
- hpss_SetNextIORquestID
 - Client API, [252](#)
 - File Data, [310](#)
- hpss_SiteldToName
 - Client API, [252](#)
- hpss_SiteNameTold
 - Client API, [253](#)
- hpss_Stage
 - Client API, [253](#)

- HPSS_STAGE_STATUS_ACTIVE
 - Client API, [89](#)
- HPSS_STAGE_STATUS_QUEUED
 - Client API, [89](#)
- HPSS_STAGE_STATUS_UNKNOWN
 - Client API, [89](#)
- hpss_StageBatch
 - Client API, [254](#)
- hpss_StageBatchCallBack
 - Client API, [255](#)
- hpss_StageCallBack
 - Client API, [255](#)
- hpss_StageCallBackBitfile
 - Client API, [257](#)
- hpss_Stat
 - Client API, [258](#)
 - File Attribute, [345](#)
- hpss_stat, [82](#)
- hpss_stat.h, [727](#)
- hpss_Statfs
 - Client API, [258](#)
 - HPSS Statistics, [392](#)
- hpss_statfs, [83](#)
- hpss_Statfs64
 - Client API, [259](#)
 - HPSS Statistics, [392](#)
- hpss_statfs64, [83](#)
- hpss_Statvfs
 - Client API, [259](#)
 - HPSS Statistics, [393](#)
- hpss_statvfs, [84](#)
- hpss_Statvfs64
 - Client API, [260](#)
 - HPSS Statistics, [393](#)
- hpss_statvfs64, [84](#)
- hpss_String.c, [728](#)
 - AdminStateStrings, [751](#)
 - AuthenticatorTypeStrings, [752](#)
 - AuthnMechStrings, [752](#)
 - AuthzMechStrings, [752](#)
 - DriveFlagStrings, [752](#)
 - DriveStateStrings, [753](#)
 - hpss_AdminStateString, [731](#)
 - hpss_AllocStateString, [731](#)
 - hpss_AuthenticatorTypeFromString, [732](#)
 - hpss_AuthenticatorTypeString, [732](#)
 - hpss_AuthnMechTypeFromString, [733](#)
 - hpss_AuthnMechTypeString, [733](#)
 - hpss_AuthzMechTypeFromString, [733](#)
 - hpss_AuthzMechTypeString, [734](#)
 - hpss_Bytecount32, [734](#)
 - hpss_Bytecount64, [735](#)
 - hpss_DriveFlagString, [735](#)
 - hpss_DriveStateString, [735](#)
 - hpss_DriveTypeString, [736](#)
 - hpss_ErrnoName, [736](#)
 - hpss_getline, [736](#)
 - hpss_HashTypeString, [737](#)
 - hpss_HexString, [737](#)
 - hpss_LabelTypeString, [738](#)
 - hpss_MediaTypeString, [738](#)
 - hpss_MountModeString, [738](#)
 - hpss_OpStateString, [739](#)
 - hpss_ProtLevelFromString, [739](#)
 - hpss_ProtLevelString, [739](#)
 - hpss_PVLActivityStateString, [740](#)
 - hpss_PVLJobStateString, [740](#)
 - hpss_PVRTypeString, [741](#)
 - hpss_ServerTypeString, [741](#)
 - hpss_string_cmp, [741](#)
 - hpss_string_fromCstr, [742](#)
 - hpss_stringbuf_clear, [742](#)
 - hpss_stringbuf_cmp, [742](#)
 - hpss_stringbuf_delBackward, [743](#)
 - hpss_stringbuf_delForward, [743](#)
 - hpss_stringbuf_free, [744](#)
 - hpss_stringbuf_fromCstr, [744](#)
 - hpss_stringbuf_fromFile, [745](#)
 - hpss_stringbuf_fromString, [745](#)
 - hpss_stringbuf_grow, [746](#)
 - hpss_stringbuf_insert, [746](#)
 - hpss_stringbuf_printf, [747](#)
 - hpss_StringClean, [747](#)
 - hpss_StringsAreEqual, [747](#)
 - hpss_time_string, [748](#)
 - hpss-Token_Add, [748](#)
 - hpss-Tokenize, [749](#)
 - hpss_UsageStateString, [749](#)
 - LabelTypeStrings, [753](#)
 - NumAdminStateStrings, [753](#)
 - NumAuthenticatorTypeStrings, [753](#)
 - NumAuthnMechStrings, [754](#)
 - NumAuthzMechStrings, [754](#)
 - NumDriveFlagStrings, [754](#)
 - NumDriveStateStrings, [754](#)
 - NumErrnoStrings, [754](#)
 - NumLabelTypeStrings, [755](#)
 - NumOpStateStrings, [755](#)
 - NumProtLevelStrings, [755](#)
 - NumPVLActivityStateStrings, [755](#)
 - NumPVLJobStateStrings, [755](#)
 - NumPVRTypeStrings, [756](#)
 - NumServerSubtypeStrings, [756](#)
 - NumServerTypeStrings, [756](#)
 - NumUsageStateStrings, [756](#)
 - OpStateStrings, [756](#)
 - ProtLevelStrings, [757](#)
 - PVLActivityStateStrings, [757](#)
 - PVLJobStateStrings, [757](#)
 - PVRTypeStrings, [758](#)
 - s32_and_map, [750](#)
 - s32_eq_map, [750](#)
 - ServerSubtypeStrings, [758](#)
 - ServerTypeStrings, [758](#)
 - u32_and_map, [751](#)
 - u32_eq_map, [751](#)

- UsageStateStrings, [759](#)
- hpss_string_cmp
 - hpss_String.c, [741](#)
- hpss_string_fromCStr
 - hpss_String.c, [742](#)
- hpss_stringbuf_clear
 - hpss_String.c, [742](#)
- hpss_stringbuf_cmp
 - hpss_String.c, [742](#)
- hpss_stringbuf_delBackward
 - hpss_String.c, [743](#)
- hpss_stringbuf_delForward
 - hpss_String.c, [743](#)
- hpss_stringbuf_free
 - hpss_String.c, [744](#)
- hpss_stringbuf_fromCStr
 - hpss_String.c, [744](#)
- hpss_stringbuf_fromFile
 - hpss_String.c, [745](#)
- hpss_stringbuf_fromString
 - hpss_String.c, [745](#)
- hpss_stringbuf_grow
 - hpss_String.c, [746](#)
- hpss_stringbuf_insert
 - hpss_String.c, [746](#)
- hpss_stringbuf_printf
 - hpss_String.c, [747](#)
- hpss_StringClean
 - hpss_String.c, [747](#)
- hpss_StringsAreEqual
 - hpss_String.c, [747](#)
- hpss_Symlink
 - Client API, [260](#)
 - File Name, [352](#)
- hpss_SymlinkHandle
 - Client API, [261](#)
 - File Name, [352](#)
- hpss_ThreadCleanUp
 - Client API, [262](#)
 - Client API Control, [386](#)
- hpss_time_string
 - hpss_String.c, [748](#)
- hpss_Token_Add
 - hpss_String.c, [748](#)
- hpss_Tokenize
 - hpss_String.c, [749](#)
- hpss_Truncate
 - Client API, [262](#)
 - File Data, [311](#)
- hpss_TruncateHandle
 - Client API, [263](#)
 - File Data, [312](#)
- hpss_Umask
 - Client API, [264](#)
- hpss_Undelete
 - Client API, [264](#)
 - Trashcan, [443](#)
- hpss_undelete_flags_t
 - Client API, [93](#)
- HPSS_UNDELETE_NONE
 - Client API, [93](#)
- HPSS_UNDELETE_OVERWRITE
 - Client API, [93](#)
- HPSS_UNDELETE_OVERWRITE_AND_RESTORE
 - Client API, [93](#)
- HPSS_UNDELETE_RESTORE_TIME
 - Client API, [93](#)
- hpss_UndeleteHandle
 - Client API, [265](#)
 - Trashcan, [444](#)
- hpss_Unlink
 - Client API, [265](#)
 - File Open, Create, and Close, [294](#)
- hpss_UnlinkHandle
 - Client API, [266](#)
 - File Open, Create, and Close, [296](#)
- hpss_UnlinkHandleImmediate
 - Client API, [267](#)
 - File Open, Create, and Close, [297](#)
- hpss_UnlinkImmediate
 - Client API, [268](#)
 - File Open, Create, and Close, [298](#)
- hpss_UpdateACL
 - ACLs, [359](#)
 - Client API, [268](#)
- hpss_UpdateACLHandle
 - ACLs, [360](#)
 - Client API, [269](#)
- hpss_UsageStateString
 - hpss_String.c, [749](#)
- hpss_userattr, [84](#)
- hpss_userattr_list, [85](#)
- hpss_userattr_t
 - Client API, [91](#)
- hpss_UserAttrCloseCursor
 - Client API, [270](#)
 - User-defined Attributes, [408](#)
- hpss_UserAttrDeleteAttrHandle
 - Client API, [270](#)
 - User-defined Attributes, [409](#)
- hpss_UserAttrDeleteAttrs
 - Client API, [271](#)
 - User-defined Attributes, [409](#)
- hpss_UserAttrGetAttrHandle
 - Client API, [272](#)
 - User-defined Attributes, [410](#)
- hpss_UserAttrGetAttrs
 - Client API, [273](#)
 - User-defined Attributes, [411](#)
- hpss_UserAttrGetObj
 - User-defined Attributes, [412](#)
- hpss_UserAttrGetObjQuery
 - User-defined Attributes, [412](#)
- hpss_UserAttrGetXML
 - User-defined Attributes, [413](#)
- hpss_UserAttrGetXMLObj

- User-defined Attributes, [414](#)
- hpss_UserAttrListAttrHandle
 - Client API, [273](#)
- hpss_UserAttrListAttrs
 - Client API, [274](#)
 - User-defined Attributes, [415](#)
- hpss_UserAttrReadObj
 - Client API, [275](#)
 - User-defined Attributes, [416](#)
- hpss_UserAttrReadXML
 - Client API, [276](#)
 - User-defined Attributes, [416](#)
- hpss_UserAttrReadXMLObj
 - Client API, [276](#)
 - User-defined Attributes, [417](#)
- hpss_UserAttrSetAttrHandle
 - Client API, [277](#)
 - User-defined Attributes, [417](#)
- hpss_UserAttrSetAttrs
 - Client API, [278](#)
 - User-defined Attributes, [418](#)
- hpss_UserAttrXQueryGet
 - Client API, [278](#)
 - User-defined Attributes, [419](#)
- hpss_UserAttrXQueryGetHandle
 - Client API, [279](#)
 - User-defined Attributes, [420](#)
- hpss_UserAttrXQueryUpdate
 - Client API, [280](#)
 - User-defined Attributes, [420](#)
- hpss_UserAttrXQueryUpdateHandle
 - Client API, [280](#)
 - User-defined Attributes, [421](#)
- hpss_Utime
 - Client API, [281](#)
 - File Attribute, [345](#)
- hpss_ValidXPath
 - HPSS XML Interface, [491](#)
- hpss_vattr.h, [759](#)
- hpss_Write
 - Client API, [282](#)
 - File Data, [312](#)
- hpss_WriteList
 - Client API, [282](#)
- hpss_xml.c, [760](#)
- hpss_xml.h, [761](#)
- hpss_XMLtoAttr
 - HPSS XML Interface, [492](#)
- hpss_XPathBranch
 - HPSS XML Interface, [492](#)
- hpss_XPathLeaf
 - HPSS XML Interface, [493](#)
- hpss_XPathRBranch
 - HPSS XML Interface, [494](#)
- hpss_XPathRoot
 - HPSS XML Interface, [494](#)
- hpss_XPathtoXML
 - HPSS XML Interface, [495](#)
- hpsscfg_config_api.c, [761](#)
- hpsscfgx_cfg_functions.c, [762](#)
- hpsscfgx_ConfGetClientInterfaces
 - HPSS Client API Extensions, [507](#)
- hpsscfgx_ConfParse
 - HPSS Client API Extensions, [508](#)
 - HPSS Config File API, [546](#)
- hpsscfgx_ConfSetDirPaths
 - HPSS Client API Extensions, [509](#)
- hpsscfgx_ConfSetFileName
 - HPSS Client API Extensions, [509](#)
- hpsscfgx_DisposeConf
 - Main, [514](#)
- HPSSCFGX_ENUM
 - Parsers, [517](#)
- hpsscfgx_FormatMessage
 - Main, [514](#)
- hpsscfgx_GetClientInterfaces.c, [763](#)
- hpsscfgx_hpssconf.c, [763](#)
- hpsscfgx_LookupHostName
 - HPSS Client API Extensions, [510](#)
- hpsscfgx_NetoptFindEntry
 - HPSS Client API Extensions, [511](#)
- hpsscfgx_NetoptGetWriteSize
 - HPSS Client API Extensions, [511](#)
- hpsscfgx_NetoptSetSock
 - HPSS Client API Extensions, [512](#)
- hpsscfgx_ParseTemplate
 - Main, [515](#)
- hpsscfgx_pattern_match.c, [764](#)
- MAX_PATTERN_BUF, [765](#)
- hpsscfgx_PatternMatch
 - HPSS Client API Extensions, [512](#)
- hpsscfgx_ReadConf
 - Main, [515](#)
- hpsscfgx_ReadConfTemplate
 - Main, [516](#)
- hpsscfgx_restricted_ports.c, [765](#)
- HPSSCFGX_STRING
 - Parsers, [517](#)
- inc64m
 - 64-bit Math Library, [501](#)
- IRBTREE_BLACK
 - Data Containers, [454](#)
- IRBTREE_RED
 - Data Containers, [454](#)
- LabelTypeStrings
 - hpss_String.c, [753](#)
- le64m
 - 64-bit Math Library, [501](#)
- low32m
 - 64-bit Math Library, [501](#)
- lt64m
 - 64-bit Math Library, [501](#)
- Main, [514](#)
 - hpsscfgx_DisposeConf, [514](#)

- hpsscfx_FormatMessage, [514](#)
 - hpsscfx_ParseTemplate, [515](#)
 - hpsscfx_ReadConf, [515](#)
 - hpsscfx_ReadConfTemplate, [516](#)
- MAX_LOAD_FACTOR
 - Data Containers, [454](#)
- MAX_PATTERN_BUF
 - hpsscfx_pattern_match.c, [765](#)
- MAX_XPATH_ELEM_SIZE
 - Client API, [89](#)
- MAX_XPATH_SIZE
 - Client API, [89](#)
- mem64m
 - 64-bit Math Library, [502](#)
- MIN_RESTRICTED_PORT
 - HPSS Client API Extensions, [506](#)
- mod2x64m
 - 64-bit Math Library, [502](#)
- mod64_3m
 - 64-bit Math Library, [502](#)
- mod64m
 - 64-bit Math Library, [502](#)
- mul64_3m
 - 64-bit Math Library, [502](#)
- mul64m
 - 64-bit Math Library, [503](#)
- NAMESPEC_GROUP
 - Client API, [93](#)
- NAMESPEC_REALM
 - Client API, [93](#)
- NAMESPEC_SKIP
 - Client API, [93](#)
- namespec_type_t
 - Client API, [93](#)
- NAMESPEC_USER
 - Client API, [93](#)
- neq64m
 - 64-bit Math Library, [503](#)
- neqz64m
 - 64-bit Math Library, [503](#)
- NormalizePath
 - api_path.c, [656](#)
- not64m
 - 64-bit Math Library, [503](#)
- NOTRUNC_CLEAR
 - Client API, [94](#)
- notrunc_flag_t
 - Client API, [94](#)
- NOTRUNC_SET
 - Client API, [94](#)
- NumAdminStateStrings
 - hpss_String.c, [753](#)
- NumAuthenticatorTypeStrings
 - hpss_String.c, [753](#)
- NumAuthnMechStrings
 - hpss_String.c, [754](#)
- NumAuthzMechStrings
 - hpss_String.c, [754](#)
- NumDriveFlagStrings
 - hpss_String.c, [754](#)
- NumDriveStateStrings
 - hpss_String.c, [754](#)
- NumErrnoStrings
 - hpss_String.c, [754](#)
- NumLabelTypeStrings
 - hpss_String.c, [755](#)
- NumOpStateStrings
 - hpss_String.c, [755](#)
- NumProtLevelStrings
 - hpss_String.c, [755](#)
- NumPVLActivityStateStrings
 - hpss_String.c, [755](#)
- NumPVLJobStateStrings
 - hpss_String.c, [755](#)
- NumPVRTypeStrings
 - hpss_String.c, [756](#)
- NumServerSubtypeStrings
 - hpss_String.c, [756](#)
- NumServerTypeStrings
 - hpss_String.c, [756](#)
- NumUsageStateStrings
 - hpss_String.c, [756](#)
- OpStateStrings
 - hpss_String.c, [756](#)
- or64m
 - 64-bit Math Library, [503](#)
- orbit64m
 - 64-bit Math Library, [503](#)
- Parallel I/O, [425](#)
 - hpss_PIOEnd, [426](#)
 - hpss_PIOExecute, [427](#)
 - hpss_PIOExecuteItem, [427](#)
 - hpss_PIOExportGrp, [428](#)
 - hpss_PIOFinalize, [429](#)
 - hpss_PIOGetPartError, [429](#)
 - hpss_PIOGetRequestID, [430](#)
 - hpss_PIOImportGrp, [430](#)
 - hpss_PIORegister, [431](#)
 - hpss_PIOStart, [432](#)
- Parsers, [516](#)
 - HPSSCFGX_ENUM, [517](#)
 - HPSSCFGX_STRING, [517](#)
- PIOInit
 - api_pio.c, [658](#)
- POSIX APIs, [519](#)
 - hpss_CreateWithHints, [519](#)
 - hpss_OpenWithHints, [520](#)
 - hpss_PosixCreate, [521](#)
 - hpss_PosixFstat, [521](#)
 - hpss_PosixLseek, [522](#)
 - hpss_PosixLstat, [522](#)
 - hpss_PosixOpen, [523](#)
 - hpss_PosixStat, [523](#)
- ProtLevelStrings
 - hpss_String.c, [757](#)

- PURGE_LOCK
 - Client API, [94](#)
- PURGE_ON_MIGRATE_CLEAR
 - Client API, [95](#)
- PURGE_ON_MIGRATE_SET
 - Client API, [95](#)
- PURGE_UNLOCK
 - Client API, [94](#)
- purgelock_flag_t
 - Client API, [94](#)
- purgeonmigrate_flag_t
 - Client API, [94](#)
- PVLActivityStateStrings
 - hpss_String.c, [757](#)
- PVLJobStateStrings
 - hpss_String.c, [757](#)
- PVRTypeStrings
 - hpss_String.c, [758](#)
- s32_and_map
 - hpss_String.c, [750](#)
- s32_eq_map
 - hpss_String.c, [750](#)
- ServerSubtypeStrings
 - hpss_String.c, [758](#)
- ServerTypeStrings
 - hpss_String.c, [758](#)
- shl64_3m
 - 64-bit Math Library, [504](#)
- shl64_ipm
 - 64-bit Math Library, [504](#)
- shl64m
 - 64-bit Math Library, [504](#)
- shr64_3m
 - 64-bit Math Library, [504](#)
- shr64_ipm
 - 64-bit Math Library, [504](#)
- shr64m
 - 64-bit Math Library, [505](#)
- Simplified Parallel I/O, [432](#)
 - hpss_PIOInitCtx, [433](#)
 - hpss_PIOReadFile, [434](#)
 - hpss_PIOReadFileMulti, [434](#)
 - hpss_PIOResetCtx, [435](#)
 - hpss_PIOSetBlockSize, [435](#)
 - hpss_PIOSetDefaultCtx, [436](#)
 - hpss_PIOSetHandleGaps, [436](#)
 - hpss_PIOSetOffset, [437](#)
 - hpss_PIOSetStripeWidth, [437](#)
 - hpss_PIOSetUseAPIHost, [438](#)
 - hpss_PIOWriteFile, [438](#)
 - hpss_PIOWriteFileMulti, [439](#)
- Special, [518](#)
- stale_thresh_cmp_factor
 - api_cache.c, [592](#)
- sub64_3m
 - 64-bit Math Library, [505](#)
- sub64m
 - 64-bit Math Library, [505](#)
- subsys_stats, [85](#)
- SUPER_PURGE_LOCK
 - Client API, [94](#)
- SUPER_PURGE_UNLOCK
 - Client API, [94](#)
- thresh_cmp
 - api_cache.c, [593](#)
- Trashcan, [440](#)
 - hpss_GetTrashcan, [440](#)
 - hpss_GetTrashcanHandle, [441](#)
 - hpss_GetTrashSettings, [442](#)
 - hpss_ReadTrash, [442](#)
 - hpss_Undelete, [443](#)
 - hpss_UndeleteHandle, [444](#)
- u32_and_map
 - hpss_String.c, [751](#)
- u32_eq_map
 - hpss_String.c, [751](#)
- u_signed64.h, [766](#)
 - CS_DEC64_LEN, [767](#)
- UDA_API_VALUE
 - Client API, [89](#)
- UDA_API_XML
 - Client API, [90](#)
- UsageStateStrings
 - hpss_String.c, [759](#)
- User-defined Attributes, [407](#)
 - hpss_UserAttrCloseCursor, [408](#)
 - hpss_UserAttrDeleteAttrHandle, [409](#)
 - hpss_UserAttrDeleteAttrs, [409](#)
 - hpss_UserAttrGetAttrHandle, [410](#)
 - hpss_UserAttrGetAttrs, [411](#)
 - hpss_UserAttrGetObj, [412](#)
 - hpss_UserAttrGetObjQuery, [412](#)
 - hpss_UserAttrGetXML, [413](#)
 - hpss_UserAttrGetXMLObj, [414](#)
 - hpss_UserAttrListAttrs, [415](#)
 - hpss_UserAttrReadObj, [416](#)
 - hpss_UserAttrReadXML, [416](#)
 - hpss_UserAttrReadXMLObj, [417](#)
 - hpss_UserAttrSetAttrHandle, [417](#)
 - hpss_UserAttrSetAttrs, [418](#)
 - hpss_UserAttrXQueryGet, [419](#)
 - hpss_UserAttrXQueryGetHandle, [420](#)
 - hpss_UserAttrXQueryUpdate, [420](#)
 - hpss_UserAttrXQueryUpdateHandle, [421](#)
- userkeycompare
 - api_cache.c, [593](#)
- Validators, [518](#)
- Working Directory, [376](#)
 - hpss_Chdir, [376](#)
 - hpss_Chroot, [377](#)
 - hpss_Getcwd, [378](#)
 - hpss_GetThreadUcred, [378](#)
- XML_ATTR

Client API, [90](#)
XML_NO_ATTR
Client API, [90](#)