This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, The AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, The AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, The AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, The AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, The AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.
Contents

Preface .....................................................................................................................................................9

Extended Library Functions, Volume 4 ........................................................................................................13
acl_check(3SEC) ........................................................................................................................................14
aclcheck(3SEC) ........................................................................................................................................15
acl_free(3SEC) ........................................................................................................................................17
acl_get(3SEC) ........................................................................................................................................18
aclsornt(3SEC) ........................................................................................................................................20
acl_strip(3SEC) ........................................................................................................................................21
acltomode(3SEC) ....................................................................................................................................22
acl_totext(3SEC) ....................................................................................................................................24
acltotext(3SEC) ....................................................................................................................................29
acl_trivial(3SEC) ....................................................................................................................................31
blcompare(3TSOL) ..................................................................................................................................32
blminmax(3TSOL) ....................................................................................................................................33
bltcolor(3TSOL) .....................................................................................................................................34
bltos(3TSOL) ..........................................................................................................................................36
btohex(3TSOL) .......................................................................................................................................39
cpl_complete_word(3TECLA) ................................................................................................................41
ef_expand_file(3TECLA) .......................................................................................................................47
getpathbylabel(3TSOL) ..........................................................................................................................51
getplabel(3TSOL) ....................................................................................................................................53
getuserrange(3TSOL) ..............................................................................................................................54
getzonelabelbyid(3TSOL) .........................................................................................................................56
getzoneroootbyid(3TSOL) .........................................................................................................................58
gl_get_line(3TECLA) ............................................................................................................................60
gl_io_mode(3TECLA) ..............................................................................................................................87
hextob(3TSOL) ......................................................................................................................................94
<table>
<thead>
<tr>
<th>Function</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>labelclipping</td>
<td>3TSOL</td>
<td>95</td>
</tr>
<tr>
<td>label_to_str</td>
<td>3TSOL</td>
<td>97</td>
</tr>
<tr>
<td>libtecla_version</td>
<td>3TECLA</td>
<td>99</td>
</tr>
<tr>
<td>media_findname</td>
<td>3VOLMGT</td>
<td>100</td>
</tr>
<tr>
<td>media_getattr</td>
<td>3VOLMGT</td>
<td>102</td>
</tr>
<tr>
<td>media_getid</td>
<td>3VOLMGT</td>
<td>104</td>
</tr>
<tr>
<td>m_label</td>
<td>3TSOL</td>
<td>106</td>
</tr>
<tr>
<td>pca_lookup_file</td>
<td>3TECLA</td>
<td>108</td>
</tr>
<tr>
<td>sbtos</td>
<td>3TSOL</td>
<td>113</td>
</tr>
<tr>
<td>scf_decoration_create</td>
<td>3SCF</td>
<td>115</td>
</tr>
<tr>
<td>scf_entry_create</td>
<td>3SCF</td>
<td>121</td>
</tr>
<tr>
<td>scf_error</td>
<td>3SCF</td>
<td>123</td>
</tr>
<tr>
<td>scf_handle_create</td>
<td>3SCF</td>
<td>125</td>
</tr>
<tr>
<td>scf_handle_decode_fmri</td>
<td>3SCF</td>
<td>128</td>
</tr>
<tr>
<td>scf_instance_create</td>
<td>3SCF</td>
<td>131</td>
</tr>
<tr>
<td>scf_iter_create</td>
<td>3SCF</td>
<td>136</td>
</tr>
<tr>
<td>scf_limit</td>
<td>3SCF</td>
<td>142</td>
</tr>
<tr>
<td>scf_pg_create</td>
<td>3SCF</td>
<td>143</td>
</tr>
<tr>
<td>scf_property_create</td>
<td>3SCF</td>
<td>150</td>
</tr>
<tr>
<td>scf_scope_create</td>
<td>3SCF</td>
<td>154</td>
</tr>
<tr>
<td>scf_service_create</td>
<td>3SCF</td>
<td>156</td>
</tr>
<tr>
<td>scf_service_delcust</td>
<td>3SCF</td>
<td>160</td>
</tr>
<tr>
<td>scf_service_is_masked</td>
<td>3SCF</td>
<td>162</td>
</tr>
<tr>
<td>scf_simple_prop_get</td>
<td>3SCF</td>
<td>164</td>
</tr>
<tr>
<td>scf_simple_walk_instances</td>
<td>3SCF</td>
<td>171</td>
</tr>
<tr>
<td>scf_snaplevel_create</td>
<td>3SCF</td>
<td>172</td>
</tr>
<tr>
<td>scf_snapshot_create</td>
<td>3SCF</td>
<td>175</td>
</tr>
<tr>
<td>scf_tmpl_pg_create</td>
<td>3SCF</td>
<td>179</td>
</tr>
<tr>
<td>scf_tmpl_pg_name</td>
<td>3SCF</td>
<td>182</td>
</tr>
<tr>
<td>scf_tmpl_prop_create</td>
<td>3SCF</td>
<td>185</td>
</tr>
<tr>
<td>scf_tmpl_prop_name</td>
<td>3SCF</td>
<td>187</td>
</tr>
<tr>
<td>scf_tmpl_validate_fmri</td>
<td>3SCF</td>
<td>194</td>
</tr>
<tr>
<td>scf_transaction_create</td>
<td>3SCF</td>
<td>200</td>
</tr>
<tr>
<td>scf_value_create</td>
<td>3SCF</td>
<td>206</td>
</tr>
<tr>
<td>setflabel</td>
<td>3TSOL</td>
<td>211</td>
</tr>
<tr>
<td>smf_enable_instance</td>
<td>3SCF</td>
<td>214</td>
</tr>
<tr>
<td>Function Name</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>smf_method_exit</td>
<td>218</td>
<td></td>
</tr>
<tr>
<td>smf_notify_set_params</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>srpt_SetDefaultState</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>srpt_SetTargetState</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>SSAgentIsAlive</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>SSAOidCmp</td>
<td>228</td>
<td></td>
</tr>
<tr>
<td>SSAStringCpy</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>stmfAddToHostGroup</td>
<td>231</td>
<td></td>
</tr>
<tr>
<td>stmfAddToHostGroupList</td>
<td>232</td>
<td></td>
</tr>
<tr>
<td>stmfAddToTargetGroup</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>stmfAddViewEntry</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>stmfAddViewEntryList</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>stmfCheckHostGroupInUse</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>stmfCheckTargetGroupInUse</td>
<td>239</td>
<td></td>
</tr>
<tr>
<td>stmfClearProviderData</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>stmfCreateHostGroup</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>stmfCreateLu</td>
<td>242</td>
<td></td>
</tr>
<tr>
<td>stmfCreateLuResource</td>
<td>244</td>
<td></td>
</tr>
<tr>
<td>stmfCreateTargetGroup</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>stmfDeleteHostGroup</td>
<td>246</td>
<td></td>
</tr>
<tr>
<td>stmfDeleteLu</td>
<td>247</td>
<td></td>
</tr>
<tr>
<td>stmfDeleteTargetGroup</td>
<td>248</td>
<td></td>
</tr>
<tr>
<td>stmfDestroyProxyDoor</td>
<td>249</td>
<td></td>
</tr>
<tr>
<td>stmfDevidFromIscsiName</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>stmfDevidFromWwn</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td>stmfFreeLuResource</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>stmfFreeMemory</td>
<td>253</td>
<td></td>
</tr>
<tr>
<td>stmfFreeViewResourceList</td>
<td>254</td>
<td></td>
</tr>
<tr>
<td>stmfGetAluaState</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>stmfGetHostGroupList</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>stmfGetHostGroupMembers</td>
<td>257</td>
<td></td>
</tr>
<tr>
<td>stmfGetLogicalUnitList</td>
<td>258</td>
<td></td>
</tr>
<tr>
<td>stmfGetLogicalUnitProperties</td>
<td>259</td>
<td></td>
</tr>
<tr>
<td>stmfGetLuResource</td>
<td>260</td>
<td></td>
</tr>
<tr>
<td>stmfGetPersistMethod</td>
<td>261</td>
<td></td>
</tr>
<tr>
<td>stmfGetProviderData</td>
<td>262</td>
<td></td>
</tr>
</tbody>
</table>
stmfGetProviderDataProt(3STMF) ................................................................. 263
stmfGetState(3STMF) .................................................................................. 264
stmfGetStmfProp(3STMF) ............................................................................ 265
stmfGetTargetGroupList(3STMF) ................................................................. 266
stmfGetTargetGroupMembers(3STMF) ......................................................... 267
stmfGetTargetList(3STMF) ........................................................................... 268
stmfGetTargetProperties(3STMF) ............................................................... 269
stmfGetViewEntryList(3STMF) .................................................................... 270
stmfGetViewLuNumberList(3STMF) ............................................................. 271
stmfGetViewProp(3STMF) .......................................................................... 272
stmfGetViewResourceList(3STMF) ............................................................... 273
stmfImportLu(3STMF) .................................................................................. 274
stmfInitProxyDoor(3STMF) .......................................................................... 275
stmfLuStandby(3STMF) .............................................................................. 276
stmfModifyLu(3STMF) ................................................................................ 277
stmfOfflineLogicalUnit(3STMF) ................................................................. 278
stmfOfflineLogicalUnit(3STMF) ................................................................. 279
stmfOfflineTarget(3STMF) .......................................................................... 280
stmfOnlineLogicalUnit(3STMF) ................................................................... 281
stmfOnlineTarget(3STMF) .......................................................................... 282
stmfPostProxyMsg(3STMF) ......................................................................... 283
stmfRemoveFromHostGroup(3STMF) ......................................................... 284
stmfRemoveFromTargetGroup(3STMF) ....................................................... 285
stmfRemoveViewEntry(3STMF) .................................................................... 286
stmfSetAluaState(3STMF) ........................................................................... 287
stmfSetLuProp(3STMF) ............................................................................... 288
stmfSetPersistMethod(3STMF) ................................................................. 289
stmfSetProviderData(3STMF) ...................................................................... 290
stmfSetProviderDataProt(3STMF) ............................................................. 291
stmfSetStmfProp(3STMF) .......................................................................... 292
stmfValidateView(3STMF) ......................................................................... 293
stobl(3TSOL) ......................................................................................... 294
str_to_label(3TSOL) .................................................................................. 295
sysevent_bind_handle(3SYSEVENT) ......................................................... 296
sysevent_free(3SYSEVENT) ........................................................................ 297
sysevent_get_attr_list(3SYSEVENT) ....................................................... 298
sysevent_get_class_name(3SYSEVENT) ................................................... 299
sysevent_get_attr_list(3SYSEVENT) ....................................................... 300
sysevent_get_class_name(3SYSEVENT) ................................................... 301
# Contents

zs_pset(3ZONESTAT) .................................................................................................................... 358  
zs_pset_zone(3ZONESTAT) ......................................................................................................... 361  
zs_resource(3ZONESTAT) ........................................................................................................... 364  
zs_usage(3ZONESTAT) .................................................................................................................. 367  
zs_zone(3ZONESTAT) .................................................................................................................... 371
Both novice users and those familiar with the SunOS operating system can use online man pages to obtain information about the system and its features. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 7 describes various special files that refer to specific hardware peripherals and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
- Section 9E describes the DDI (Device Driver Interface)/DKI (Driver/Kernel Interface), DDI-only, and DKI-only entry-point routines a developer can include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report,
there is no BUGS section. See the Intro pages for more information and detail about each section, and man(1) for more information about man pages in general.

**NAME**

This section gives the names of the commands or functions documented, followed by a brief description of what they do.

**SYNOPSIS**

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

- **[]** Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.
- **...** Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "filename...".
- **|** Separator. Only one of the arguments separated by this character can be specified at a time.
- **{}** Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

**PROTOCOL**

This section occurs only in subsection 3R to indicate the protocol description file.

**DESCRIPTION**

This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.

**IOCTL**

This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the ioctl(2) system call is called ioctl and generates its own heading. ioctl calls for a specific device are listed alphabetically (on the man page for that specific device).
ioctl calls are used for a particular class of devices all of which have an io ending, such as mtio(7I).

OPTIONS
This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

OPERANDS
This section lists the command operands and describes how they affect the actions of the command.

OUTPUT
This section describes the output – standard output, standard error, or output files – generated by the command.

RETURN VALUES
If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or −1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.

ERRORS
On failure, most functions place an error code in the global variable errno indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE
This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality:

Commands
Modifiers
Variables
Expressions
Input Grammar

EXAMPLES
This section provides examples of usage or of how to use a command or function. Wherever possible a complete
example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.

**ENVIRONMENT VARIABLES**

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

**EXIT STATUS**

This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.

**FILES**

This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

**ATTRIBUTES**

This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See `attributes(5)` for more information.

**SEE ALSO**

This section lists references to other man pages, in-house documentation, and outside publications.

**DIAGNOSTICS**

This section lists diagnostic messages with a brief explanation of the condition causing the error.

**WARNINGS**

This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.

**NOTES**

This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

**BUGS**

This section describes known bugs and, wherever possible, suggests workarounds.
REFERENCE

Extended Library Functions, Volume 4
The acl_check() function checks the validity of an ACL pointed to by aclp. The isdir argument checks the validity of an ACL that will be applied to a directory. The ACL can be either a POSIX draft ACL as supported by UFS or NFSv4 ACL as supported by ZFS or NFSv4.

When the function verifies a POSIX draft ACL, the rules followed are described in aclcheck(3SEC). For NFSv4 ACL, the ACL is verified against the following rules:

- The inheritance flags are valid.
- The ACL must have at least one ACL entry and no more than MAX_ACL_ENTRIES.
- The permission field contains only supported permissions.
- The entry type is valid.
- The flag fields contain only valid flags as supported by NFSv4/ZFS.

If any of the above rules are violated, the function fails with errno set to EINVAL.

If the ACL is valid, acl_check() returns 0. Otherwise errno is set to EINVAL and the return value is set to one of the following:

- EACL_INHERIT_ERROR: There are invalid inheritance flags specified.
- EACL_FLAGS_ERROR: There are invalid flags specified on the ACL that don’t map to supported flags in NFSv4/ZFS ACL model.
- EACL_ENTRY_ERROR: The ACL contains an unknown value in the type field.
- EACL_MEM_ERROR: The system cannot allocate any memory.
- EACL_INHERIT_NOTDIR: Inheritance flags are only allowed for ACLs on directories.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also acl(2), aclcheck(3SEC), aclsort(3SEC), acl(5), attributes(5)
Name  aclcheck – check the validity of an ACL

Synopsis  cc { flag... } file... -lsec { library... }
          #include <sys/acl.h>

          int aclcheck(aclent_t *aclbufp, int nentries, int *which);

Description  The aclcheck() function checks the validity of an ACL pointed to by aclbufp. The nentries argument is the number of entries contained in the buffer. The which parameter returns the index of the first entry that is invalid.

The function verifies that an ACL pointed to by aclbufp is valid according to the following rules:

- There must be exactly one GROUP_OBJ ACL entry.
- There must be exactly one USER_OBJ ACL entry.
- There must be exactly one OTHER_OBJ ACL entry.
- If there are any GROUP ACL entries, then the group ID in each group ACL entry must be unique.
- If there are any USER ACL entries, then the user ID in each user ACL entry must be unique.
- If there are any GROUP or USER ACL entries, then there must be exactly one CLASS_OBJ (ACL mask) entry.
- If there are any default ACL entries, then the following apply:
  - There must be exactly one default GROUP_OBJ ACL entry.
  - There must be exactly one default OTHER_OBJ ACL entry.
  - There must be exactly one default USER_OBJ ACL entry.
  - If there are any DEF_GROUP entries, then the group ID in each DEF_GROUP ACL entry must be unique.
  - If there are any DEF_USER entries, then the user ID in each DEF_USER ACL entry must be unique.
  - If there are any DEF_GROUP or DEF_USER entries, then there must be exactly one DEF_CLASS_OBJ (default ACL mask) entry.
  - If any of the above rules are violated, then the function fails with errno set to EINVAL.

Return Values  If the ACL is valid, aclcheck() will return 0. Otherwise errno is set to EINVAL and return code is set to one of the following:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRP_ERROR</td>
<td>There is more than one GROUP_OBJ or DEF_GROUP_OBJ ACL entry.</td>
</tr>
<tr>
<td>USER_ERROR</td>
<td>There is more than one USER_OBJ or DEF_USER_OBJ ACL entry.</td>
</tr>
<tr>
<td>CLASS_ERROR</td>
<td>There is more than one CLASS_OBJ (ACL mask) or DEF_CLASS_OBJ (default ACL mask) entry.</td>
</tr>
</tbody>
</table>
OTHER_ERROR: There is more than one OTHER_OBJ or DEF_OTHER_OBJ ACL entry.

DUPLICATE_ERROR: Duplicate entries of USER, GROUP, DEF_USER, or DEF_GROUP.

ENTRY_ERROR: The entry type is invalid.

MISS_ERROR: Missing an entry. The which parameter returns −1 in this case.

MEM_ERROR: The system cannot allocate any memory. The which parameter returns −1 in this case.

Attributes: See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTEVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also: acl(2), aclsort(3SEC), attributes(5)
acl_free - free memory associated with an acl_t structure

Synopsis

```c
cc [ flag... ] file... -lsec [ library... ]
#include <sys/acl.h>

void acl_free(acl_t *aclp);
```

Description

The `acl_free()` function frees memory allocated for the acl_t structure pointed to by the `aclp` argument.

Return Values

The `acl_free()` function does not return a value.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`acl_get(3SEC), acl(5), attributes(5)`
acl_get(3SEC)

Name acl_get, facl_get, acl_set, facl_set – get or set a file’s Access Control List (ACL)

Synopsis cc [ flag... ] file... -lssec [ library... ]
#include <sys/acl.h>

int acl_get(const char *path, int flag, acl_t **aclp);
int facl_get(int fd, int flag, acl_t **aclp);
int acl_set(const char *path, acl_t *aclp);
int facl_set(int fd, acl_t *aclp);

Description The acl_get() and facl_get() functions retrieve an Access Control List (ACL) of a file whose name is given by path or referenced by the open file descriptor fd. The flag argument specifies whether a trivial ACL should be retrieved. When the flag argument is ACL_NO_TRIVIAL, only ACLs that are not trivial will be retrieved. The ACL is returned in the aclp argument.

The acl_set() and facl_set() functions are used for setting an ACL of a file whose name is given by path or referenced by the open file descriptor fd. The aclp argument specifies the ACL to set.

The acl_get() and acl_set() functions support multiple types of ACLs. When possible, the acl_set() function translates an ACL to the target file’s style of ACL. Currently this is only possible when translating from a POSIX-draft ACL such as on UFS to a file system that supports NFSv4 ACL semantics such as ZFS or NFSv4.

The caller is responsible for freeing the returned acl_t structure using acl_free(3SEC).

Return Values Upon successful completion, acl_get() and facl_get() return 0 and aclp is non-NULL. The aclp argument can be NULL after successful completion if the file had a trivial ACL and the flag argument was ACL_NO_TRIVIAL. Otherwise, -1 is returned and errno is set to indicate the error.

Upon successful completion, acl_set() and facl_set() return 0. Otherwise, -1 is returned and errno is set to indicate the error.

Errors These functions will fail if:

EACCES The caller does not have access to a component of path.
EIO A disk I/O error has occurred while retrieving the ACL.
ENOENT A component of the path does not exist.
ENOSYS The file system does not support ACLs.
ENOTSUP The ACL supplied could not be translated to an NFSv4 ACL.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  chmod(1), acl(2), acl_free(3SEC), acl(5), attributes(5)
The `aclsort()` function sorts the contents of the ACL buffer as follows:

- Entries will be in the order USER_OBJ, USER, GROUP_OBJ, GROUP, CLASS_OBJ (ACL mask), OTHER_OBJ, DEF_USER_OBJ, DEF_USER, DEF_GROUP_OBJ, DEF_GROUP, DEF_CLASS_OBJ (default ACL mask), and DEF_OTHER_OBJ.
- Entries of type USER, GROUP, DEF_USER, and DEF_GROUP will be sorted in increasing order by ID.

The `aclsort()` function will succeed if all of the following are true:

- There is exactly one entry each of type USER_OBJ, GROUP_OBJ, CLASS_OBJ (ACL mask), and OTHER_OBJ.
- There is exactly one entry each of type DEF_USER_OBJ, DEF_GROUP_OBJ, DEF_CLASS_OBJ (default ACL mask), and DEF_OTHER_OBJ if there are any default entries.
- Entries of type USER, GROUP, DEF_USER, or DEF_GROUP may not contain duplicate entries. A duplicate entry is one of the same type containing the same numeric ID.

Upon successful completion, the function returns 0. Otherwise, it returns -1.

See also `acl(2), aclcheck(3SEC), attributes(5)`
Name acl_strip – remove all ACLs from a file

Synopsis cc { flag... } file... -lsec { library... }
#include <sys/acl.h>

int acl_strip(const char *path, uid_t uid, gid_t gid, mode_t mode);

Description The acl_strip() function removes all ACLs from a file and replaces them with a trivial ACL based on the mode argument. After replacing the ACL, the owner and group of the file are set to the values specified by the uid and gid arguments.

Return Values Upon successful completion, acl_strip() returns 0. Otherwise it returns -1 and sets errno to indicate the error.

Errors The acl_strip() function will fail if:

EACCES Search permission is denied on a component of the path prefix of path.
EFAULT The path argument points to an illegal address.
EINVAL The uid or gid argument is out of range.
EIO A disk I/O error has occurred while storing or retrieving the ACL.
ELOOP A loop exists in symbolic links encountered during the resolution of the path argument.
ENAMETOOLONG The length of the path argument exceeds {PATH_MAX}, or the length of a path component exceeds {NAME_MAX} while _POSIX_NO_TRUNC is in effect.
ENOENT A component of path does not exist.
ENOTDIR A component of the prefix of path is not a directory.
EPERM The effective user ID does not match the owner of the file and the process does not have appropriate privileges.
EROFS The file system is mounted read-only.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also acl_get(3SEC), acl_trivial(3SEC), acl(5), attributes(5)
The `acltomode()` function converts an ACL pointed to by `aclbufp` into the permission bits buffer pointed to by `modep`. If the USER_OBJ ACL entry, GROUP_OBJ ACL entry, or the OTHER_OBJ ACL entry cannot be found in the ACL buffer, then the function fails with `errno` set to EINVAL.

The USER_OBJ ACL entry permission bits are copied to the file owner class bits in the permission bits buffer. The OTHER_OBJ ACL entry permission bits are copied to the file other class bits in the permission bits buffer. If there is a CLASS_OBJ (ACL mask) entry, the CLASS_OBJ ACL entry permission bits are copied to the file group class bits in the permission bits buffer. Otherwise, the GROUP_OBJ ACL entry permission bits are copied to the file group class bits in the permission bits buffer.

The `aclfrommode()` function converts the permission bits pointed to by `modep` into an ACL pointed to by `aclbufp`. If the USER_OBJ ACL entry, GROUP_OBJ ACL entry, or the OTHER_OBJ ACL entry cannot be found in the ACL buffer, the function fails with `errno` set to EINVAL.

The file owner class bits from the permission bits buffer are copied to the USER_OBJ ACL entry. The file other class bits from the permission bits buffer are copied to the OTHER_OBJ ACL entry. If there is a CLASS_OBJ (ACL mask) entry, the file group class bits from the permission bits buffer are copied to the CLASS_OBJ ACL entry, and the GROUP_OBJ ACL entry is not modified. Otherwise, the file group class bits from the permission bits buffer are copied to the GROUP_OBJ ACL entry.

The `nentries` argument represents the number of ACL entries in the buffer pointed to by `aclbufp`.

Upon successful completion, the function returns 0. Otherwise, it returns −1 and sets `errno` to indicate the error.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
<td></td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
<td></td>
</tr>
</tbody>
</table>
See Also acl(2), attributes(5)
acl_totext(3SEC)

Name  acl_totext, acl_fromtext – convert internal representation to or from external representation

Synopsis  cc [ flag...] file... -lsec [ library... ]
#include <sys/acl.h>

char *acl_totext(acl_t *aclp, int flags);
int acl_fromtext(char *acltextp, acl_t **aclp);

Description  The acl_totext() function converts an internal ACL representation pointed to by aclp into an external ACL representation. The memory for the external text string is obtained using malloc(3C). The caller is responsible for freeing the memory upon completion.

The format of the external ACL is controlled by the flags argument. Values for flags are constructed by a bitwise-inclusive-OR of flags from the following list, defined in <sys/acl.h>.

ACL_COMPACT_FMT  For NFSv4 ACLs, the ACL entries will be formatted using the compact ACL format detailed in ls(1) for the -V option.

ACL_APPEND_ID  Append the uid or gid for additional user or group entries. This flag is used to construct ACL entries in a manner that is suitable for archive utilities such as tar(1). When the ACL is translated from the external format to internal representation using acl_fromtext(), the appended ID will be used to populate the uid or gid field of the ACL entry when the user or group name does not exist on the host system. The appended id will be ignored when the user or group name does exist on the system.

ACL_SID_FMT  For NFSv4 ACLs, the ACL entries for user or group entries will use the usersid or groupsid format when the “id” field in the ACL entry is an ephemeral uid or gid. The raw sid format will only be used when the “id” cannot be resolved to a windows name.

The acl_fromtext() function converts an external ACL representation pointed to by acltextp into an internal ACL representation. The memory for the list of ACL entries is obtained using malloc(3C). The caller is responsible for freeing the memory upon completion. Depending on type of ACLs a file system supports, one of two external representations are possible. For POSIX draft file systems such as ufs, the external representation is described in acltotext(3SEC). The external ACL representation For NFSv4–style ACLs is detailed as follows.

Each acl_entry contains one ACL entry. The external representation of an ACL entry contains three, four or five colon separated fields. The first field contains the ACL entry type. The entry type keywords are defined as:

everyone@  This ACL entry specifies the access granted to any user or group that does not match any previous ACL entry.
group This ACL entry with a GID specifies the access granted to a additional group of the object.

group@ This ACL entry with no GID specified in the ACL entry field specifies the access granted to the owning group of the object.

groupsid This ACL entry with a SID or Windows name specifies the access granted to a Windows group. This type of entry is for a SMB server created file.

owner@ This ACL entry with no UID specified in the ACL entry field specifies the access granted to the owner of the object.

sid This ACL entry with a SID or Windows name when the entry could be either a group or a user.

user This ACL entry with a UID specifies the access granted to a additional user of the object.

usersid This ACL entry with a SID or Windows name specifies the access granted to a Windows user. This type of entry is for a SMB server created file.

The second field contains the ACL entry ID, and is used only for user or group ACL entries. This field is not used for owner@, group@, or everyone@ entries.

uid This field contains a user-name or user-ID. If the user-name cannot be resolved to a UID, then the entry is assumed to be a numeric UID.

gid This field contains a group-name or group-ID. If the group-name can’t be resolved to a GID, then the entry is assumed to be a numeric GID.

The third field contains the discretionary access permissions. The format of the permissions depends on whether ACL_COMPACT_FMT is specified. When the flags field does not request ACL_COMPACT_FMT, the following format is used with a forward slash (/) separating the permissions.

add_file Add a file to a directory.

add_subdirectory Add a subdirectory.

append Append data.

delete Delete.

delete_child Delete child.

execute Execute permission.

list_directory List a directory.

read_acl Read ACL.

read_data Read permission.
read_attributes Read attributes.
read_xattr Read named attributes.
synchronize Synchronize.
write_acl Write ACL.
write_attributes Write attributes.
write_data Write permission.
write_owner Write owner.
write_xattr Write named attributes.

This format allows permissions to be specified as, for example:
read_data/read_xattr/read_attributes.

When ACL_COMPACT_FMT is specified, the permissions consist of 14 unique letters. A hyphen (-) character is used to indicate that the permission at that position is not specified.

a read attributes
A write attributes
c read ACL
C write ACL
d delete
D delete child
o write owner
p append
r read_data
R read named attributes
s synchronize
w write_data
W write named attributes
x execute

This format allows compact permissions to be represented as, for example: rw- -d-a- - - - - -

The fourth field is optional when ACL_COMPACT_FMT is not specified, in which case the field will be present only when the ACL entry has inheritance flags set. The following is the list of inheritance flags separated by a slash (/) character.
dir_inherit ACE_DIRECTORY_INHERIT_ACE
file_inherit ACE_FILE_INHERIT_ACE
inherit_only ACE_INHERIT_ONLY_ACE
no_propagate ACE_NO_PROPAGATE_INHERIT_ACE

When ACL_COMPACT_FMT is specified the inheritance will always be present and is represented as positional arguments. A hyphen (-) character is used to indicate that the inheritance flag at that position is not specified.

d dir_inherit
f file_inherit
F failed access (not currently supported)
i inherit_only
n no_propagate
S successful access (not currently supported)

The fifth field contains the type of the ACE (allow or deny):
allow The mask specified in field three should be allowed.
deny The mask specified in field three should be denied.

**Return Values**

Upon successful completion, the acl_totext() function returns a pointer to a text string. Otherwise, it returns NULL.

Upon successful completion, the acl_fromtext() function returns 0. Otherwise, the return value is set to one of the following:

- **EACL_FIELD_NOT_BLANK**: A field that should be blank is not blank.
- **EACL_FLAGS_ERROR**: An invalid ACL flag was specified.
- **EACL_INHERIT_ERROR**: An invalid inheritance field was specified.
- **EACL_INVALID_ACCESS_TYPE**: An invalid access type was specified.
- **EACL_INVALID_STR**: The string is NULL.
- **EACL_INVALID_USER_GROUP**: The required user or group name not found.
- **EACL_MISSING_FIELDS**: The ACL needs more fields to be specified.
- **EACL_PERM_MASK_ERROR**: The permission mask is invalid.
- **EACL_UNKNOWN_DATA**: Unknown data was found in the ACL.
**Examples**  
**EXAMPLE 1**  
Examples of permissions when ACL_COMPACT_FMT is not specified.

user:joe:read_data/write_data:file_inherit/dir_inherit:allow  
owner@:read_acl:allow,user:tom:read_data:file_inherit/inherit_only:deny

**EXAMPLE 2**  
Examples of permissions when ACL_COMPACT_FMT is specified.

user:joe:rw----------:fd----:allow  
owner@:----------c------:allow,user:tom:r------------:f-i---:deny

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
ls(1), tar(1), acl(2), malloc(3C), aclfromtext(3SEC), acl(5), attributes(5)
Name acltotext, aclfromtext – convert internal representation to or from external representation

Synopsis

c { flag... } file... -lsec { library... }
#include <sys/acl.h>

char *acltotext(aclent_t *aclbufp, int aclcnt);
aclent_t *aclfromtext(char *acltextp, int *aclcnt);

Description

The acltotext() function converts an internal ACL representation pointed to by aclbufp into an external ACL representation. The space for the external text string is obtained using malloc(3C). The caller is responsible for freeing the space upon completion.

The aclfromtext() function converts an external ACL representation pointed to by acltextp into an internal ACL representation. The space for the list of ACL entries is obtained using malloc(3C). The caller is responsible for freeing the space upon completion. The aclcnt argument indicates the number of ACL entries found.

An external ACL representation is defined as follows:

<acl_entry>,<acl_entry>...

Each <acl_entry> contains one ACL entry. The external representation of an ACL entry contains two or three colon-separated fields. The first field contains the ACL entry tag type. The entry type keywords are defined as:

user This ACL entry with no UID specified in the ACL entry ID field specifies the access granted to the owner of the object. Otherwise, this ACL entry specifies the access granted to a specific user-name or user-id number.

group This ACL entry with no GID specified in the ACL entry ID field specifies the access granted to the owning group of the object. Otherwise, this ACL entry specifies the access granted to a specific group-name or group-id number.

other This ACL entry specifies the access granted to any user or group that does not match any other ACL entry.

mask This ACL entry specifies the maximum access granted to user or group entries.

default:user This ACL entry with no uid specified in the ACL entry ID field specifies the default access granted to the owner of the object. Otherwise, this ACL entry specifies the default access granted to a specific user-name or user-ID number.

default:group This ACL entry with no gid specified in the ACL entry ID field specifies the default access granted to the owning group of the object. Otherwise, this ACL entry specifies the default access granted to a specific group-name or group-ID number.
default:other  This ACL entry specifies the default access for other entry.
default:mask  This ACL entry specifies the default access for mask entry.

The second field contains the ACL entry ID, as follows:
uid     This field specifies a user-name, or user-ID if there is no user-name associated with the user-ID number.
gid     This field specifies a group-name, or group-ID if there is no group-name associated with the group-ID number.
empty   This field is used by the user and group ACL entry types.

The third field contains the following symbolic discretionary access permissions:
r     read permission
w     write permission
x     execute/search permission
−     no access

Return Values  Upon successful completion, the acltotext() function returns a pointer to a text string. Otherwise, it returns NULL.

Upon successful completion, the aclfromtext() function returns a pointer to a list of ACL entries. Otherwise, it returns NULL.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also  acl(2), malloc(3C), attributes(5)
The `acl_trivial()` function is used to determine whether a file has a trivial ACL. Whether an ACL is trivial depends on the type of the ACL. A POSIX draft ACL is nontrivial if it has greater than `MIN_ACL_ENTRIES`. An NFSv4/ZFS-style ACL is nontrivial if it either has entries other than `owner@`, `group@`, and `everyone@`, has inheritance flags set, or is not ordered in a manner that meets POSIX access control requirements.

Upon successful completion, `acl_trivial()` returns 0 if the file's ACL is trivial and 1 if the file's ACL is not trivial. If it could not be determined whether a file's ACL is trivial, -1 is returned and `errno` is set to indicate the error.

The `acl_trivial()` function will fail if:

- `EACCES` A file's ACL could not be read.
- `ENOENT` A component of `path` does not name an existing file or `path` is an empty string.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `acl(5), attributes(5)`
#include <tsol/label.h>

int blequal(const m_label_t *label1, const m_label_t *label2);
int bldominates(const m_label_t *label1, const m_label_t *label2);
int blstrictdom(const m_label_t *label1, const m_label_t *label2);
int blinrange(const m_label_t *label, const brange_t *range);

These functions compare binary labels for meeting a particular condition.

The blequal() function compares two labels for equality.

The bldominates() function compares label label1 for dominance over label label2.

The blstrictdom() function compares label label1 for strict dominance over label label2.

The blinrange() function compares label label for dominance over range→lower_bound and range→upper_bound for dominance over level label.

These functions return non-zero if their respective conditions are met, otherwise zero is returned.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also getplabel(3TSOL), label_to_str(3TSOL), libtsol(3LIB), ucred_getlabel(3C), label_encodings(4), attributes(5), labels(5)

“Determining the Relationship Between Two Labels” in Trusted Extensions Developer’s Guide

Notes The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
The `blmaximum()` function replaces the contents of label `maximum_label` with the least upper bound of the labels `maximum_label` and `bounding_label`. This is the least label that dominates both of the original labels.

The `blminimum()` function replaces the contents of label `minimum_label` with the greatest lower bound of the labels `minimum_label` and `bounding_label`. This is the greatest label that is dominated by both of the original labels.
bltocolor(3TSOL)

**Name**
bltocolor, bltocolor_r – get character-coded color name of label

**Synopsis**
```c
cc [flag...] file... -ltsol [library...]
```
```
#include <tsol/label.h>

char *bltocolor(const m_label_t *label);
char *bltocolor_r(const m_label_t *label, const int size,
                   char *color_name);
```

**Description**
The `bltocolor()` and `bltocolor_r()` functions get the character-coded color name associated with the binary label `label`.

The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to get color names of labels that dominate the current process's sensitivity label.

**Return Values**
The `bltocolor()` function returns a pointer to a statically allocated string that contains the character-coded color name specified for the `label` or returns `(char *)0` if, for any reason, no character-coded color name is available for this binary label.

The `bltocolor_r()` function returns a pointer to the `color_name` string which contains the character-coded color name specified for the `label` or returns `(char *)0` if, for any reason, no character-coded color name is available for this binary label. `color_name` must provide for a string of at least `size` characters.

**Files**
```
/etc/security/tsol/label_encodings
```

The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

These functions are obsolete and retained for ease of porting. They might be removed in a future Solaris Trusted Extensions release. Use the `label_to_str(3TSOL)` function instead.

The `bltocolor()` function returns a pointer to a statically allocated string. Subsequent calls to it will overwrite that string with a new character-coded color name. It is not MT-Safe. The `bltocolor_r()` function should be used in multithreaded applications.

**See Also**
`label_to_str(3TSOL), libtsol(3LIB), attributes(5)`
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.

If `label` includes a specified word or words, the character-coded color name associated with the first word specified in the label encodings file is returned. Otherwise, if no character-coded color name is specified for `label`, the first character-coded color name specified in the label encodings file with the same classification as the binary label is returned.
bltos(3TSOL)

**Name**
bltos, bsltos, bcleartos – translate binary labels to character coded labels

**Synopsis**
```
cc [flag...] file... -ltso [library...]
#include <tsol/label.h>

int bsltos(const m_label_t *label, char **string,
const int str_len, const int flags);
int bcleartos(const m_label_t *label, char **string,
const int str_len, const int flags);
```

**Description**
These functions translate binary labels into strings controlled by the value of the `flags` parameter.

The `bltos()` function translates a binary sensitivity label into a string. The applicable `flags` are `LONG_CLASSIFICATION` or `SHORT_CLASSIFICATION`, `LONG_WORDS` or `SHORT_WORDS`, `VIEW_EXTERNAL` or `VIEW_INTERNAL`, and `NO_CLASSIFICATION`. A `flags` value 0 is equivalent to `(SHORT_CLASSIFICATION | LONG_WORDS)`.

The `bcleartos()` function translates a binary clearance into a string. The applicable `flags` are `LONG_CLASSIFICATION` or `SHORT_CLASSIFICATION`, `LONG_WORDS` or `SHORT_WORDS`, `VIEW_EXTERNAL` or `VIEW_INTERNAL`, and `NO_CLASSIFICATION`. A `flags` value 0 is equivalent to `(SHORT_CLASSIFICATION | LONG_WORDS)`. The translation of a clearance might not be the same as the translation of a sensitivity label. These functions use different `label_encodings` file tables that might contain different words and constraints.

The calling process must have `PRIV_SYS_TRANS_LABEL` in its set of effective privileges to perform label translation on labels that dominate the current process's sensitivity label.

The generic form of an output character-coded label is:
```
CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6
```

Capital letters are used to display all `CLASSIFICATION` names and `WORDS`. The ’ ’ (space) character separates classifications and words from other words in all character-coded labels except where multiple words that require the same `PREFIX` or `SUFFIX` are present, in which case the multiple words are separated from each other by the ’/’ (slash) character.

The `string` argument can point to either a pointer to pre-allocated memory, or the value `(char *)0`. If `string` points to a pointer to pre-allocated memory, then `str_len` indicates the size of that memory. If `string` points to the value `(char *)0`, memory is allocated using `malloc()` to contain the translated character-coded labels. The translated `label` is copied into allocated or pre-allocated memory.

The `flags` argument is 0 or the logical sum of the following:
```
LONG_WORDS Translate using long names of words defined in `label`
```
Translate using short names of words defined in *label*. If no short name is defined in the *label_encodings* file for a word, the long name is used.

Translate using long name of classification defined in *label*.

Translate using short name of classification defined in *label*.

Translate only access-related entries defined in information label *label*.

Translate ADMIN_LOW and ADMIN_HIGH labels to the lowest and highest labels defined in the *label_encodings* file.

Translate ADMIN_LOW and ADMIN_HIGH labels to the admin_low and admin_high name strings specified in the *label_encodings* file. If no strings are specified, the strings "ADMIN_LOW" and "ADMIN_HIGH" are used.

Do not translate classification defined in *label*.

Process Attributes

If the VIEW_EXTERNAL or VIEW_INTERNAL flags are not specified, translation of ADMIN_LOW and ADMIN_HIGH labels is controlled by the label view process attribute flags. If no label view process attribute flags are defined, their translation is controlled by the label view configured in the *label_encodings* file. A value of External specifies that ADMIN_LOW and ADMIN_HIGH labels are mapped to the lowest and highest labels defined in the *label_encodings* file. A value of Internal specifies that the ADMIN_LOW and ADMIN_HIGH labels are translated to the admin_low and admin_high name strings specified in the *label_encodings* file. If no such names are specified, the strings "ADMIN_LOW" and "ADMIN_HIGH" are used.

Return Values

Upon successful completion, the `bsltos()` and `bcleartos()` functions return the length of the character-coded label, including the NULL terminator.

If the label is not of the valid defined required type, if the label is not dominated by the process sensitivity label and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges, or if the label_encodings file is inaccessible, these functions return −1.

If memory cannot be allocated for the return string or if the pre-allocated return string memory is insufficient to hold the string, these functions return 0. The value of the pre-allocated string is set to the NULL string (*string[0]='\00';*).

Files

`/etc/security/tsol/label_encodings`

The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

Attributes

See attributes(5) for descriptions of the following attributes:
The `bsltos()` and `bcleartos()` functions are Obsolete. Use the `label_to_str(3TSOL)` function instead.

**Attributes**

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

**See Also**

`free(3C), label_to_str(3TSOL), libtsol(3LIB), malloc(3C), label_encodings(4), attributes(5)`

**Notes**

The functionality described on this manual page is available only if the system is configured with Trusted Extensions.

If memory is allocated by these functions, the caller must free the memory with `free(3C)` when the memory is no longer in use.
btohex(3TSOL)

Name  btohex, bsltoh, bcleartoh, bsltoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal

Synopsis  cc [flag...] file... -ltsol [library...]

#include <tsol/label.h>

char *bsltoh(const m_label_t *label);
char *bcleartoh(const m_label_t *clearance);
char *bsltoh_r(const m_label_t *label, char *hex);
char *bcleartoh_r(const m_label_t *clearance, char *hex);
char *h_alloc(const unsigned char type);
void h_free(char *hex);

Description  These functions convert binary labels into hexadecimal strings that represent the internal value.

The bsltoh() and bsltoh_r() functions convert a binary sensitivity label into a string of the form:

[0xsensitivity_label_hexadecimal_value]

The bcleartoh() and bcleartoh_r() functions convert a binary clearance into a string of the form:

0xclearance_hexadecimal_value

The h_alloc() function allocates memory for the hexadecimal value type for use by bsltoh_r() and bcleartoh_r().

Valid values for type are:

SUN_SL_ID  label is a binary sensitivity label.
SUN_CLR_ID label is a binary clearance.

The h_free() function frees memory allocated by h_alloc().

Return Values  These functions return a pointer to a string that contains the result of the translation, or (char *)0 if the parameter is not of the required type.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>
The bsltoh(), bcleartoh(), bsltoh_r(), bcleartoh_r(), h_alloc(), and h_free() functions are Obsolete. Use the label_to_str(3TSOL) function instead.

The bsltoh() and bcleartoh() functions share the same statically allocated string storage. They are not MT-Safe. Subsequent calls to any of these functions will overwrite that string with the newly translated string. The bsltoh_r() and bcleartoh_r() functions should be used in multithreaded applications.

See Also: atohexlabel(1M), hextoalabel(1M), label_to_str(3TSOL), libtsol(3LIB), attributes(5), labels(5)

Notes: The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
The **cpl_complete_word()** function is part of the **libtecla(3LIB)** library. It is usually called behind the scenes by **gl_get_line(3TECLA)**, but can also be called separately.

Given an input line containing an incomplete word to be completed, it calls a user-provided callback function (or the provided file-completion callback function) to look up all possible completion suffixes for that word. The callback function is expected to look backward in the line, starting from the specified cursor position, to find the start of the word to be completed, then to look up all possible completions of that word and record them, one at a time, by calling **cpl_add_completion()**.
The `new_WordCompletion()` function creates the resources used by the `cpl_complete_word()` function. In particular, it maintains the memory that is used to return the results of calling `cpl_complete_word()`.

The `del_WordCompletion()` function deletes the resources that were returned by a previous call to `new_WordCompletion()`. It always returns `NULL` (that is, a deleted object). It takes no action if the `cpl` argument is `NULL`.

The callback functions that look up possible completions should be defined with the `CPL_MATCH_FN()` macro, which is defined in `<libtecla.h>`. Functions of this type are called by `cpl_complete_word()`, and all of the arguments of the callback are those that were passed to said function. In particular, the `line` argument contains the input line containing the word to be completed, and `word_end` is the index of the character that follows the last character of the incomplete word within this string. The callback is expected to look backwards from `word_end` for the start of the incomplete word. What constitutes the start of a word clearly depends on the application, so it makes sense for the callback to take on this responsibility. For example, the built-in filename completion function looks backwards until it encounters an unescaped space or the start of the line. Having found the start of the word, the callback should then lookup all possible completions of this word, and record each completion with separate calls to `cpl_add_completion()`. If the callback needs access to an application-specific symbol table, it can pass it and any other data that it needs using the `data` argument. This removes any need for global variables.

The callback function should return 0 if no errors occur. On failure it should return 1 and register a terse description of the error by calling `cpl_record_error()`.

The last error message recorded by calling `cpl_record_error()` can subsequently be queried by calling `cpl_last_error()`.

The `cpl_add_completion()` function is called zero or more times by the completion callback function to record each possible completion in the specified `WordCompletion` object. These completions are subsequently returned by `cpl_complete_word()`. The `cpl`, `line`, and `word_end` arguments should be those that were passed to the callback function. The `word_start` argument should be the index within the input line string of the start of the word that is being completed. This should equal `word_end` if a zero-length string is being completed. The `suffix` argument is the string that would have to be appended to the incomplete word to complete it. If this needs any quoting (for example, the addition of backslashes before special characters) to be valid within the displayed input line, this should be included. A copy of the suffix string is allocated internally, so there is no need to maintain your copy of the string after `cpl_add_completion()` returns.

In the array of possible completions that the `cpl_complete_word()` function returns, the suffix recorded by `cpl_add_completion()` is listed along with the concatenation of this suffix with the word that lies between `word_start` and `word_end` in the input line.
The type_suffix argument specifies an optional string to be appended to the completion if it is displayed as part of a list of completions by cpl_list_completions. The intention is that this indicate to the user the type of each completion. For example, the file completion function places a directory separator after completions that are directories, to indicate their nature to the user. Similarly, if the completion were a function, you could indicate this to the user by setting type_suffix to “()”. Note that the type_suffix string is not copied, so if the argument is not a literal string between speech marks, be sure that the string remains valid for at least as long as the results of cpl_complete_word() are needed.

The cont_suffix argument is a continuation suffix to append to the completed word in the input line if this is the only completion. This is something that is not part of the completion itself, but that gives the user an indication about how they might continue to extend the token. For example, the file-completion callback function adds a directory separator if the completed word is a directory. If the completed word were a function name, you could similarly aid the user by arranging for an open parenthesis to be appended.

The cpl_complete_word() is normally called behind the scenes by gl_get_line(3TECLA), but can also be called separately if you separately allocate a WordCompletion object. It performs word completion, as described at the beginning of this section. Its first argument is a resource object previously returned by new_WordCompletion(). The line argument is the input line string, containing the word to be completed. The word_end argument contains the index of the character in the input line, that just follows the last character of the word to be completed. When called by gl_get_line(), this is the character over which the user pressed TAB. The match_fn argument is the function pointer of the callback function which will lookup possible completions of the word, as described above, and the data argument provides a way for the application to pass arbitrary data to the callback function.

If no errors occur, the cpl_complete_word() function returns a pointer to a CplMatches container, as defined below. This container is allocated as part of the cpl object that was passed to cpl_complete_word(), and will thus change on each call which uses the same cpl argument.

```c
typedef struct {
    char *completion;  /* A matching completion */
    /* string */
    char *suffix;  /* The part of the */
                /* completion string which */
                /* would have to be */
                /* appended to complete the */
                /* original word. */
    const char *type_suffix; /* A suffix to be added when */
                /* listing completions, to */
                /* indicate the type of the */
                /* completion. */
} CplMatch;

typedef struct {
```
null

If an error occurs during completion, cpl_complete_word() returns NULL. A description of the error can be acquired by calling the cpl_last_error() function.

The cpl_last_error() function returns a terse description of the error which occurred on the last call to cpl_complete_word() or cpl_add_completion().

As a convenience, the return value of the last call to cpl_complete_word() can be recalled at a later time by calling cpl_recall_matches(). If cpl_complete_word() returned NULL, so will cpl_recall_matches().

When the cpl_complete_word() function returns multiple possible completions, the cpl_list_completions() function can be called upon to list them, suitably arranged across the available width of the terminal. It arranges for the displayed columns of completions to all have the same width, set by the longest completion. It also appends the type_suffix strings that were recorded with each completion, thus indicating their types to the user.

By default the gl_get_line() function, passes the CPL_MATCH_FN(cps_file_completions) completion callback function to cpl_complete_word(). This function can also be used separately, either by sending it to cpl_complete_word(), or by calling it directly from your own completion callback function.

#define CPL_MATCH_FN(fn) int (fn)(WordCompletion *cpl, 
   void *data, const char *line, 
   int word_end)

typedef CPL_MATCH_FN(CplMatchFn);

CPL_MATCH_FN(cps_file_completions);

Certain aspects of the behavior of this callback can be changed via its data argument. If you are happy with its default behavior you can pass NULL in this argument. Otherwise it should be a pointer to a CplFileConf object, previously allocated by calling new_CplFileConf().
CplFileConf objects encapsulate the configuration parameters of CPL_FILE_COMPLETIONS(). These parameters, which start out with default values, can be changed by calling the accessor functions described below.

By default, the CPL_FILE_COMPLETIONS() callback function searches backwards for the start of the filename being completed, looking for the first unescaped space or the start of the input line. If you wish to specify a different location, call CFC_FILE_START() with the index at which the filename starts in the input line. Passing start_index=-1 reenables the default behavior.

By default, when CPL_FILE_COMPLETIONS() looks at a filename in the input line, each lone backslash in the input line is interpreted as being a special character which removes any special significance of the character which follows it, such as a space which should be taken as part of the filename rather than delimiting the start of the filename. These backslashes are thus ignored while looking for completions, and subsequently added before spaces, tabs and literal backslashes in the list of completions. To have unescaped backslashes treated as normal characters, call CFC_LITERAL_ESCAPES() with a non-zero value in its literal argument.

By default, CPL_FILE_COMPLETIONS() reports all files whose names start with the prefix that is being completed. If you only want a selected subset of these files to be reported in the list of completions, you can arrange this by providing a callback function which takes the full pathname of a file, and returns 0 if the file should be ignored, or 1 if the file should be included in the list of completions. To register such a function for use by CPL_FILE_COMPLETIONS(), call CFC_SET_CHECK_FN(), and pass it a pointer to the function, together with a pointer to any data that you would like passed to this callback whenever it is called. Your callback can make its decisions based on any property of the file, such as the filename itself, whether the file is readable, writable or executable, or even based on what the file contains.

```
#define CPL_CHECK_FN(fn) int (fn)(void *data, \  
  const char *pathname)

typedef CPL_CHECK_FN(CplCheckFn);

void cfc_set_check_fn(CplFileConf *cfc, CplCheckFn *chk_fn, \  
  void *chk_data);
```

The CPL_CHECK_EXE() function is a provided callback of the above type, for use with CPL_FILE_COMPLETIONS(). It returns non-zero if the filename that it is given represents a normal file that the user has execute permission to. You could use this to have CPL_FILE_COMPLETIONS() only list completions of executable files.

When you have finished with a CplFileConf variable, you can pass it to the DEL_CPLFILECONF() destructor function to reclaim its memory.

Thread Safety: It is safe to use the facilities of this module in multiple threads, provided that each thread uses a separately allocated WordCompletion object. In other words, if two threads want to do word completion, they should each call NEW_WORDCOMPLETION() to allocate their own completion objects.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  ef_expand_file(3TECLA), gl_get_line(3TECLA), libtecla(3LIB), pca_lookup_file(3TECLA), attributes(5)
The ef_expand_file() function is part of the libtecla(3LIB) library. It expands a specified filename, converting ~user/ and ~/ expressions at the start of the filename to the corresponding home directories, replacing $envvar with the value of the corresponding environment variable, and then, if there are any wildcards, matching these against existing filenames. Backslashes in the input filename are interpreted as escaping any special meanings of the characters that follow them. Only backslashes that are themselves preceded by backslashes are preserved in the expanded filename.

In the presence of wildcards, the returned list of filenames includes only the names of existing files which match the wildcards. Otherwise, the original filename is returned after expansion of tilde and dollar expressions, and the result is not checked against existing files. This mimics the file-globbing behavior of the UNIX tcsh shell.

The supported wildcards and their meanings are:

- `*` Match any sequence of zero or more characters.
- `?` Match any single character.
- `[chars]` Match any single character that appears in `chars`. If `chars` contains an expression of the form a-b, then any character between a and b, including a and b, matches. The `-` character loses its special meaning as a range specifier when it appears at the start of the sequence of characters. The `']` character also loses its significance as the terminator of the range expression if it appears immediately after the opening `[`, at which point it is treated one of the characters of the range. If you want both `']` and `']` to be part of the range, the `']` should come first and the `']` second.
- `[^chars]` The same as `[chars]` except that it matches any single character that does not appear in `chars`.

Note that wildcards never match the initial dot in filenames that start with `.`. The initial `.` must be explicitly specified in the filename. This again mimics the globbing behavior of most file systems.
UNIX shells, and its rational is based in the fact that in UNIX, files with names that start with ‘.’ are usually hidden configuration files, which are not listed by default by the \texttt{ls(1)} command.

The \texttt{new\_ExpandFile()} function creates the resources used by the \texttt{ef\_expand\_file()} function. In particular, it maintains the memory that is used to record the array of matching file names that is returned by \texttt{ef\_expand\_file()}. This array is expanded as needed, so there is no built-in limit to the number of files that can be matched.

The \texttt{del\_ExpandFile()} function deletes the resources that were returned by a previous call to \texttt{new\_ExpandFile()}. It always returns NULL (that is, a deleted object). It does nothing if the \texttt{ef} argument is NULL.

The \texttt{ef\_expand\_file()} function performs filename expansion. Its first argument is a resource object returned by \texttt{new\_ExpandFile()}. A pointer to the start of the filename to be matched is passed by the \texttt{path} argument. This must be a normal null-terminated string, but unless a length of -1 is passed in \texttt{pathlen}, only the first \texttt{pathlen} characters will be used in the filename expansion. If the length is specified as -1, the whole of the string will be expanded. A container of the following type is returned by \texttt{ef\_expand\_file()}.

\begin{verbatim}
typedef struct {
    int exists; /* True if the files in files[] exist */
    int nfile; /* The number of files in files[] */
    char ***files; /* An array of 'nfile' filenames. */
} FileExpansion;
\end{verbatim}

The \texttt{ef\_expand\_file()} function returns a pointer to a container whose contents are the results of the expansion. If there were no wildcards in the filename, the \texttt{nfile} member will be 1, and the \texttt{exists} member should be queried if it is important to know if the expanded file currently exists. If there were wildcards, then the contained \texttt{files[]} array will contain the names of the \texttt{nfile} existing files that matched the wild-carded filename, and the \texttt{exists} member will have the value 1. Note that the returned container belongs to the specified \texttt{ef} object, and its contents will change on each call, so if you need to retain the results of more than one call to \texttt{ef\_expand\_file()}, you should either make a private copy of the returned results, or create multiple file-expansion resource objects with multiple calls to \texttt{new\_ExpandFile()}. On error, NULL is returned, and an explanation of the error can be determined by calling \texttt{ef\_last\_error(ef)}.

The \texttt{ef\_last\_error()} function returns the message which describes the error that occurred on the last call to \texttt{ef\_expand\_file()}, for the given (ExpandFile *\texttt{ef}) resource object.

The \texttt{ef\_list\_expansions()} function provides a convenient way to list the filename expansions returned by \texttt{ef\_expand\_file()}. Like the \texttt{ls} utility, it arranges the filenames into equal width columns, each column having the width of the largest file. The number of columns used is thus determined by the length of the longest filename, and the specified terminal width. Beware that filenames that are longer than the specified terminal width are
printed without being truncated, so output longer than the specified terminal width can occur. The list is written to the stdio stream specified by the fp argument.

Thread Safety It is safe to use the facilities of this module in multiple threads, provided that each thread uses a separately allocated ExpandFile object. In other words, if two threads want to do file expansion, they should each call new_ExpandFile() to allocate their own file-expansion objects.

Examples EXAMPLE 1 Use of file expansion function.

The following is a complete example of how to use the file expansion function.

```c
#include <stdio.h>
#include <libtecla.h>

int main(int argc, char *argv[]) {
    ExpandFile *ef; /* The expansion resource object */
    char *filename; /* The filename being expanded */
    FileExpansion *expn; /* The results of the expansion */
    int i;

    ef = new_ExpandFile();
    if(!ef)
        return 1;

    for(arg = *(argv++); arg; arg = *(argv++)) {
        if((expn = ef_expand_file(ef, arg, -1)) == NULL) {
            fprintf(stderr, "Error expanding %s (%s).\n", arg,
                ef_last_error(ef));
        } else {
            printf("%s matches the following files:\n", arg);
            for(i=0; i<expn->nfile; i++)
                printf(" %s\n", expn->files[i]);
        }
    }

    ef = del_ExpandFile(ef);
    return 0;
}
```

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
### ef_expand_file(3TECLA)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
cpl_complete_word(3TECLA), gl_get_line(3TECLA), libtecla(3LIB), pca_lookup_file(3TECLA), attributes(5)
getpathbylabel(3TSOL)

**Name**
getpathbylabel – return the zone pathname

**Synopsis**
```c
cc [flags...] file... -ltsol [library...]
#include <tsol/label.h>

char *getpathbylabel(const char *path, char *resolved_path,
                     size_t bufsize, const m_label_t *sl);
```

**Description**
The `getpathbylabel()` function expands all symbolic links and resolves references to `'/.'`, `'/../'`, extra `'/'` characters, and stores the zone pathname in the buffer named by `resolved_path`. The `bufsize` argument specifies the size in bytes of this buffer. The resulting path will have no symbolic links components, nor any `'./', '/../'`. This function can only be called from the global zone.

The zone pathname is relative to the sensitivity label `sl`. To specify a sensitivity label for a zone name which does not exist, the process must assert either the `PRIV_FILE_UPGRADE_SL` or `PRIV_FILE_DOWNGRADE_SL` privilege depending on whether the specified sensitivity label dominates or does not dominate the process sensitivity label.

**Return Values**
The `getpathbylabel()` function returns a pointer to the `resolved_path` on success. Otherwise it returns `NULL` and sets `errno` to indicate the error.

**Errors**
The `getpathbylabel()` function will fail if:

- **EACCES** Search permission is denied for a component of the path prefix of `path`.
- **EFAULT** `resolved_path` extends outside the process's allocated address space or beyond `bufsize` bytes.
- **EINVAL** `path` or `resolved_path` was `NULL`, current zone is not the global zone, or `sl` is invalid.
- **EIO** An I/O error occurred while reading from or writing to the file system.
- **ELOOP** Too many symbolic links were encountered in translating `path`.
- **ENAMETOOLONG** The length of the path argument exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX` (see `sysconf(3C)`) while `_POSIX_NO_TRUNC` is in effect (see `pathconf(2)`).
- **ENOENT** The named file does not exist.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
The `getpathbylabel()` function indirectly invokes the `readlink(2)` system call, and hence inherits the possibility of hanging due to inaccessible file system resources.

The functionality described on this manual page is available only if the system is configured with Trusted Extensions.

**See Also**  
`readlink(2)`, `getzoneroottbyid(3TSOL)`, `libtsol(3LIB)`, `attributes(5)`, `labels(5)`

**Warnings**  
The `getpathbylabel()` function indirectly invokes the `readlink(2)` system call, and hence inherits the possibility of hanging due to inaccessible file system resources.

**Notes**  
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
#include <tsol/label.h>

int getplabel(m_label_t *label_p);

The `getplabel()` function obtains the sensitivity label of the calling process.

Upon successful completion, `getplabel()` returns 0. Otherwise it returns -1, `label_p` is unchanged, and `errno` is set to indicate the error.

The `getplabel()` function fails and `label_p` does not refer to a valid sensitivity label if:

- `EFAULT` `label_p` points to an invalid address.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

`ucred_getlabel(3C), libtsol(3LIB), m_label_alloc(3TSOL), m_label_free(3TSOL), attributes(5)`

“Obtaining a Process Label” in *Trusted Extensions Developer’s Guide*

## Notes

The functionality described on this manual page is available only if the system is configured with Trusted Extensions.

This function returns different values for system processes than `ucred_getlabel(3C)` returns.
getuserrange – get the label range of a user

**Synopsis**

c C [flags...] file... -ltsol [library...]

    #include <tsol/label.h>

    m_range_t *getuserrange(const char *username);

**Description**
The `getuserrange()` function returns the label range of `username`. The lower bound in the range is used as the initial workspace label when a user logs into a multilevel desktop. The upper bound, or clearance, is used as an upper limit to the available labels that a user can assign to labeled workspaces.

The default value for a user's label range is specified in `label_encodings(4)`. Overriding values for individual users are specified in `user_attr(4)`.

**Return Values**
The `getuserrange()` function returns `NULL` if the memory allocation fails. Otherwise, the function returns a structure which must be freed by the caller, as follows:

    m_range_t *range;
    ... 
    m_label_free(range->lower_bound);
    m_label_free(range->upper_bound);
    free(range);

**Errors**
The `getuserrange()` function will fail if:

    ENOMEM       The physical limits of the system are exceeded by size bytes of memory which cannot be allocated.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>See below.</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

The `getuserrange()` function is Committed for systems that implement the Defense Intelligence Agency (DIA) MAC policy of `label_encodings(4)`. Other policies might exist in a future release of Trusted Extensions that might make obsolete or supplement `label_encodings`.

**See Also**
`free(3C), libtsol(3LIB), m_label_free(3TSOL), label_encodings(4), user_attr(4), attributes(5)`
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.

Notes
The `getzonelabelbyid()` function returns the mandatory access control (MAC) label of `zoneid`.

The `getzonelabelbyname()` function returns the MAC label of the zone whose name is `zonename`.

The `getzoneidbylabel()` function returns the zone ID of the primary zone whose label is `label`.

All of these functions require that the specified zone's state is at least `ZONE_IS_READY`. The zone of the calling process must dominate the specified zone's label, or the calling process must be in the global zone.

### Return Values
On successful completion, the `getzonelabelbyid()` and `getzonelabelbyname()` functions return a pointer to a sensitivity label that is allocated within these functions. To free the storage, use `m_label_free(3TSOL)`. If the zone does not exist, `NULL` is returned.

On successful completion, the `getzoneidbylabel()` function returns the zone ID with the matching label. If there is no matching zone, the function returns `-1`.

### Errors
The `getzonelabelbyid()` and `getzonelabelbyname()` functions will fail if:

- **ENOENT** The specified zone does not exist.

The `getzonelabelbyid()` function will fail if:

- **ENOENT** No zone corresponds to the specified label.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

---

**getzonelabelbyid(3TSOL)**
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.

See Also Intro(2), getzonenamebyid(3C), getzoneidbyname(3C), libtsol(3LIB), m_label_free(3TSOL), attributes(5), labels(5)

Notes The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
**Name**
getzonerootbyid, getzonerootbylabel, getzonerootbyname – map between zone root pathnames and labels

**Synopsis**
cc [flags...] file... -ltsol [library...]
#include <tsol/label.h>

char *getzonerootbyid(zoneid_t zoneid);
char *getzonerootbylabel(const m_label_t *label);
char *getzonerootbyname(const char *zonename);

**Description**
The `getzonerootbyid()` function returns the root pathname of `zoneid`.

The `getzonerootbylabel()` function returns the root pathname of the zone whose label is `label`.

The `getzonerootbyname()` function returns the root pathname of `zonename`.

All of these functions require that the specified zone's state is at least ZONE_IS_READY. The zone of the calling process must dominate the specified zone's label, or the calling process must be in the global zone. The returned pathname is relative to the root path of the caller's zone.

**Return Values**
On successful completion, the `getzonerootbyid()`, `getzonerootbylabel()`, and `getzonerootbyname()` functions return a pointer to a pathname that is allocated within these functions. To free the storage, use `free(3C)`. On failure, these functions return NULL and set `errno` to indicate the error.

**Errors**
These functions will fail if:

- **EFAULT** Invalid argument; pointer location is invalid.
- **EINVAL** `zoneid` invalid, or zone not found or not ready.
- **ENOENT** Zone does not exist.
- **ENOMEM** Unable to allocate pathname.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**
`Intro(2), free(3C), getzonenambyid(3C), libtsol(3LIB), attributes(5), labels(5)`
Notes  The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
**Name**

gl_get_line, new_GetLine, del_GetLine, gl_customize_completion, gl_change_terminal,
gl_configure_getline, gl_load_history, gl_save_history, gl_group_history, gl_show_history,
gl_watch_fd, gl_inactivity_timeout, gl_terminal_size, gl_set_term_size, gl_resize_history,
gl_limit_history, gl_clear_history, gl_toggle_history, gl_lookup_history, gl_state_of_history,
gl_range_of_history, gl_size_of_history, gl_echo_mode, gl_replace_prompt,
gl_prompt_style, gl_ignore_signal, gl_trap_signal, gl_last_signal, gl_completion_action,
gl_register_action, gl_display_text, gl_return_status, gl_error_message, gl_catch_blocked,
gl_list_signals, gl_bind_keyseq, gl_erase_terminal, gl_automatic_history, gl_append_history,
gl_query_char, gl_read_char — allow the user to compose an input line

**Synopsis**

```c
cc [ flag... ] file... -ltecla [ library... ]
#include <stdio.h>
#include <libtecla.h>

GetLine *new_GetLine(size_t linelen, size_t histlen);
GetLine *del_GetLine(GetLine *gl);
char *gl_get_line(GetLine *gl, const char *prompt,
const char *start_line, int start_pos);
int gl_query_char(GetLine *gl, const char *prompt, char defchar);
int gl_read_char(GetLine *gl);
int gl_customize_completion(GetLine *gl, void *data,
CplMatchFn *match_fn);
int gl_change_terminal(GetLine *gl, FILE *input_fp,
FILE *output_fp, const char *term);
int gl_configure_getline(GetLine *gl, const char *app_string,
const char *app_file, const char *user_file);
int gl_bind_keyseq(GetLine *gl, GlKeyOrigin origin,
const char *keyseq, const char *action);
int gl_save_history(GetLine *gl, const char *filename,
const char *comment, int max_lines);
int gl_load_history(GetLine *gl, const char *filename,
const char *comment);
int gl_watch_fd(GetLine *gl, int fd, GlFdEvent event,
GlFdEventFn *callback, void *data);
int gl_inactivity_timeout(GetLine *gl, GlTimeoutFn *callback,
void *data, unsigned long sec, unsigned long nsec);
int gl_group_history(GetLine *gl, unsigned stream);
int gl_show_history(GetLine *gl, FILE *fp, const char *fmt,
int all_groups, int max_lines);
int gl_resize_history(GetLine *gl, size_t bufsize);
```
void gl_limit_history(GetLine *gl, int max_lines);
void gl_clear_history(GetLine *gl, int all_groups);
void gl_toggle_history(GetLine *gl, int enable);

GlTerminalSize gl_terminal_size(GetLine *gl, int def_ncolumn, int def_nline);
int gl_set_term_size(GetLine *gl, int ncolumn, int nline);
int gl_lookup_history(GetLine *gl, unsigned long id, GlHistoryLine *hline);
void gl_state_of_history(GetLine *gl, GlHistoryState *state);
void gl_range_of_history(GetLine *gl, GlHistoryRange *range);
void gl_size_of_history(GetLine *gl, GlHistorySize *size);
void gl_echo_mode(GetLine *gl, int enable);
void gl_replace_prompt(GetLine *gl, const char *prompt);
void gl_prompt_style(GetLine *gl, GlPromptStyle style);
int gl_ignore_signal(GetLine *gl, int signo);
int gl_trap_signal(GetLine *gl, int signo, unsigned flags, GlAfterSignal after, int errno_value);
int gl_last_signal(GetLine *gl);

int gl_completion_action(GetLine *gl, void *data, CplMatchFn *match_fn, int list_only, const char *name, const char *keyseq);
int gl_register_action(GetLine *gl, void *data, GlActionFn *fn, const char *name, const char *keyseq);

int gl_display_text(GetLine *gl, int indentation, const char *prefix, const char *suffix, int fill_char, int def_width, int start, const char *string);

GlReturnStatus gl_return_status(GetLine *gl);
const char *gl_error_message(GetLine *gl, char *buff, size_t n);
void gl_catch_blocked(GetLine *gl);
int gl_list_signals(GetLine *gl, sigset_t *set);
int gl_append_history(GetLine *gl, const char *line);
int gl_automatic_history(GetLine *gl, int enable);
int gl_erase_terminal(GetLine *gl);
The gl_get_line() function is part of the libtecla(3LIB) library. If the user is typing at a terminal, each call prompts them for an line of input, then provides interactive editing facilities, similar to those of the UNIX tcsh shell. In addition to simple command-line editing, it supports recall of previously entered command lines, TAB completion of file names, and in-line wild-card expansion of filenames. Documentation of both the user-level command-line editing features and all user configuration options can be found on the tecla(5) manual page.

The following shows a complete example of how to use the gl_get_line() function to get input from the user:

```c
#include <stdio.h>
#include <locale.h>
#include <libtecla.h>

int main(int argc, char *argv[])
{
    char *line; /* The line that the user typed */
    GetLine *gl; /* The gl_get_line() resource object */

    setlocale(LC_CTYPE, ""); /* Adopt the user's choice */
    /* of character set. */

    gl = new_GetLine(1024, 2048);
    if(!gl)
        return 1;
    while((line=gl_get_line(gl, "$ ", NULL, -1)) != NULL &&
          strcmp(line, "exit\n") != 0 )
        printf("You typed: %s\n", line);

    gl = del_GetLine(gl);
    return 0;
}
```

In the example, first the resources needed by the gl_get_line() function are created by calling new_GetLine(). This allocates the memory used in subsequent calls to the gl_get_line() function, including the history buffer for recording previously entered lines. Then one or more lines are read from the user, until either an error occurs, or the user types exit. Then finally the resources that were allocated by new_GetLine(), are returned to the system by calling del_GetLine(). Note the use of the NULL return value of del_GetLine() to make gl NULL. This is a safety precaution. If the program subsequently attempts to pass gl to gl_get_line(), said function will complain, and return an error, instead of attempting to use the deleted resource object.

The new_GetLine() function creates the resources used by the gl_get_line() function and returns an opaque pointer to the object that contains them. The maximum length of an input line is specified by the linelen argument, and the number of bytes to allocate for storing history lines is set by the histlen argument. History lines are stored back-to-back in a single buffer of
this size. Note that this means that the number of history lines that can be stored at any given
time, depends on the lengths of the individual lines. If you want to place an upper limit on the
number of lines that can be stored, see the description of the gl_limit_history() function. If
you do not want history at all, specify histlen as zero, and no history buffer will be allocated.

On error, a message is printed to stderr and NULL is returned.

The del_GetLine() function deletes the resources that were returned by a previous call to
new_GetLine(). It always returns NULL (for example, a deleted object). It does nothing if the gl
argument is NULL.

The gl_get_line() function can be called any number of times to read input from the user.
The gl argument must have been previously returned by a call to new_GetLine(). The prompt
argument should be a normal null-terminated string, specifying the prompt to present the
user with. By default prompts are displayed literally, but if enabled with the
gl_prompt_style() function, prompts can contain directives to do underlining, switch to
and from bold fonts, or turn highlighting on and off.

If you want to specify the initial contents of the line for the user to edit, pass the desired string
with the start_line argument. You can then specify which character of this line the cursor is
initially positioned over by using the start_pos argument. This should be -1 if you want the
cursor to follow the last character of the start line. If you do not want to preload the line in this
manner, send start_line as NULL, and set start_pos to -1.

The gl_get_line() function returns a pointer to the line entered by the user, or NULL on error
or at the end of the input. The returned pointer is part of the specified gl resource object, and
thus should not be freed by the caller, or assumed to be unchanged from one call to the next.
When reading from a user at a terminal, there will always be a newline character at the end of
the returned line. When standard input is being taken from a pipe or a file, there will similarly
be a newline unless the input line was too long to store in the internal buffer. In the latter case
you should call gl_get_line() again to read the rest of the line. Note that this behavior makes
gl_get_line() similar to fgets(3C). When stdin is not connected to a terminal,
gl_get_line() simply calls fgets().

The Return Status Of
The gl_get_line() function has two possible return values: a pointer to the completed input
line, or NULL. Additional information about what caused gl_get_line() to return is available
both by inspecting errno and by calling the gl_return_status() function.

The following are the possible enumerated values returned by gl_return_status():

GLR_NEWLINE The last call to gl_get_line() successfully returned a completed input line.

GLR_BLOCKED The gl_get_line() function was in non-blocking server mode, and
returned early to avoid blocking the process while waiting for terminal I/O.
The gl_pending_io() function can be used to see what type of I/O
gl_get_line() was waiting for. See the gl_io_mode(3TECLA).
GLR_SIGNAL A signal was caught by gl_get_line() that had an after-signal disposition of GLS_ABORT. See gl_trap_signal().

GLR_TIMEOUT The inactivity timer expired while gl_get_line() was waiting for input, and the timeout callback function returned GLTO_ABORT. See gl_inactivity_timeout() for information about timeouts.

GLR_FDABORT An application I/O callback returned GLFD_ABORT. See gl_watch_fd().

GLR_EOF End of file reached. This can happen when input is coming from a file or a pipe, instead of the terminal. It also occurs if the user invokes the list-or-eof or del-char-or-list-or-eof actions at the start of a new line.

GLR_ERROR An unexpected error caused gl_get_line() to abort (consult errno and/or gl_error_message() for details.

When gl_return_status() returns GLR_ERROR and the value of errno is not sufficient to explain what happened, you can use the gl_error_message() function to request a description of the last error that occurred.

The return value of gl_error_message() is a pointer to the message that occurred. If the buff argument is NULL, this will be a pointer to a buffer within gl whose value will probably change on the next call to any function associated with gl_get_line(). Otherwise, if a non-null buff argument is provided, the error message, including a \0 terminator, will be written within the first n elements of this buffer, and the return value will be a pointer to the first element of this buffer. If the message will not fit in the provided buffer, it will be truncated to fit.

Whereas by default the prompt string that you specify is displayed literally without any special interpretation of the characters within it, the gl_prompt_style() function can be used to enable optional formatting directives within the prompt.

The style argument, which specifies the formatting style, can take any of the following values:

GL_FORMAT_PROMPT In this style, the formatting directives described below, when included in prompt strings, are interpreted as follows:

%B Display subsequent characters with a bold font.
%b Stop displaying characters with the bold font.
%F Make subsequent characters flash.
%f Turn off flashing characters.
%U Underline subsequent characters.
%u Stop underlining characters.
%P Switch to a pale (half brightness) font.
%p Stop using the pale font.
Highlight subsequent characters (also known as standout mode).

Stop highlighting characters.

Turn on reverse video.

Turn off reverse video.

Display a single % character.

For example, in this mode, a prompt string like "%UOK%u$" would display the prompt "OK $", but with the OK part underlined.

Note that although a pair of characters that starts with a % character, but does not match any of the above directives is displayed literally, if a new directive is subsequently introduced which does match, the displayed prompt will change, so it is better to always use %% to display a literal %.

Also note that not all terminals support all of these text attributes, and that some substitute a different attribute for missing ones.

GL_LITERAL_PROMPT

In this style, the prompt string is printed literally. This is the default style.

By default users have the option of configuring the behavior of gl_get_line() with a configuration file called .teclarc in their home directories. The fact that all applications share this same configuration file is both an advantage and a disadvantage. In most cases it is an advantage, since it encourages uniformity, and frees the user from having to configure each application separately. In some applications, however, this single means of configuration is a problem. This is particularly true of embedded software, where there’s no filesystem to read a configuration file from, and also in applications where a radically different choice of keybindings is needed to emulate a legacy keyboard interface. To cater for such cases, the gl_configure_getline() function allows the application to control where configuration information is read from.

The gl_configure_getline() function allows the configuration commands that would normally be read from a user’s ~/.teclarc file, to be read from any or none of, a string, an application specific configuration file, and/or a user-specific configuration file. If this function is called before the first call to gl_get_line(), the default behavior of reading ~/.teclarc on the first call to gl_get_line() is disabled, so all configurations must be achieved using the configuration sources specified with this function.

If app_string != NULL, then it is interpreted as a string containing one or more configuration commands, separated from each other in the string by embedded newline characters. If app_file != NULL then it is interpreted as the full pathname of an application-specific
configuration file. If user_file is not equal to NULL, then it is interpreted as the full path name of a user-specific configuration file, such as ~/.teclarc. For example, in the call

```c
gl_configure_getline(gl, "edit-mode vi \n nobeep",
                  "/usr/share/myapp/teclarc", "~/.teclarc");
```

The `app_string` argument causes the calling application to start in vi(1) edit-mode, instead of the default emacs mode, and turns off the use of the terminal bell by the library. It then attempts to read system-wide configuration commands from an optional file called /usr/share/myapp/teclarc, then finally reads user-specific configuration commands from an optional .teclarc file in the user's home directory. Note that the arguments are listed in ascending order of priority, with the contents of `app_string` being potentially over ridden by commands in `app_file`, and commands in `app_file` potentially being overridden by commands in `user_file`.

You can call this function as many times as needed, the results being cumulative, but note that copies of any file names specified with the `app_file` and `user_file` arguments are recorded internally for subsequent use by the read-init-files key-binding function, so if you plan to call this function multiple times, be sure that the last call specifies the filenames that you want re-read when the user requests that the configuration files be re-read.

Individual key sequences can also be bound and unbound using the `gl_bind_keyseq()` function. The `origin` argument specifies the priority of the binding, according to whom it is being established for, and must be one of the following two values.

- **GL_USER_KEY** The user requested this key-binding.
- **GL_APP_KEY** This is a default binding set by the application.

When both user and application bindings for a given key sequence have been specified, the user binding takes precedence. The application's binding is subsequently reinstated if the user's binding is later unbound with either another call to this function, or a call to `gl_configure_getline()`.

The `keyseq` argument specifies the key sequence to be bound or unbound, and is expressed in the same way as in a ~/.teclarc configuration file. The `action` argument must either be a string containing the name of the action to bind the key sequence to, or it must be NULL or "" to unbind the key sequence.

If in your application you would like to have TAB completion complete other things in addition to or instead of filenames, you can arrange this by registering an alternate completion callback function with a call to the `gl_customize_completion()` function.

The `data` argument provides a way for your application to pass arbitrary, application-specific information to the callback function. This is passed to the callback every time that it is called. It might for example point to the symbol table from which possible completions are to be sought. The `match_fn` argument specifies the callback function to be called. The `CplMatchFn`
function type is defined in <libtecla.h>, as is a CPL_MATCH_FN() macro that you can use to declare and prototype callback functions. The declaration and responsibilities of callback functions are described in depth on the cpl_complete_word(3TECLA) manual page.

The callback function is responsible for looking backwards in the input line from the point at which the user pressed TAB, to find the start of the word being completed. It then must lookup possible completions of this word, and record them one by one in the WordCompletion object that is passed to it as an argument, by calling the cpl_add_completion() function. If the callback function wants to provide filename completion in addition to its own specific completions, it has the option of itself calling the builtin filename completion callback. This also is documented on the cpl_complete_word(3TECLA) manual page.

If you would like gl_get_line() to return the current input line when a successful completion is been made, you can arrange this when you call cpl_add_completion() by making the last character of the continuation suffix a newline character. The input line will be updated to display the completion, together with any continuation suffix up to the newline character, and gl_get_line() will return this input line.

If your callback function needs to write something to the terminal, it must call gl_normal_io() before doing so. This will start a new line after the input line that is currently being edited, reinstate normal terminal I/O, and notify gl_get_line() that the input line will need to be redrawn when the callback returns.

Adding Completion Actions

In the previous section the ability to customize the behavior of the only default completion action, complete-word, was described. In this section the ability to install additional action functions, so that different types of word completion can be bound to different key sequences, is described. This is achieved by using the gl_completion_action() function.

The data and match_fn arguments are as described on the cpl_complete_word(3TECLA) manual page, and specify the callback function that should be invoked to identify possible completions. The list_only argument determines whether the action that is being defined should attempt to complete the word as far as possible in the input line before displaying any possible ambiguous completions, or whether it should simply display the list of possible completions without touching the input line. The former option is selected by specifying a value of 0, and the latter by specifying a value of 1. The name argument specifies the name by which configuration files and future invocations of this function should refer to the action. This must either be the name of an existing completion action to be changed, or be a new unused name for a new action. Finally, the keyseq argument specifies the default key sequence to bind the action to. If this is NULL, no new key sequence will be bound to the action.

Beware that in order for the user to be able to change the key sequence that is bound to actions that are installed in this manner, you shouldcall gl_completion_action() to install a given action for the first time between calling new_GetLine() and the first call to gl_get_line().
Otherwise, when the user's configuration file is read on the first call to `gl_get_line()`, the name of the your additional action will not be known, and any reference to it in the configuration file will generate an error.

As discussed for `gl_customize_completion()`, if your callback function needs to write anything to the terminal, it must call `gl_normal_io()` before doing so.

Although the built-in key-binding actions are sufficient for the needs of most applications, occasionally a specialized application may need to define one or more custom actions, bound to application-specific key sequences. For example, a sales application would benefit from having a key sequence that displayed the part name that corresponded to a part number preceding the cursor. Such a feature is clearly beyond the scope of the built-in action functions. So for such special cases, the `gl_register_action()` function is provided.

The `gl_register_action()` function lets the application register an external function, `fn`, that will thereafter be called whenever either the specified key sequence, `keyseq`, is entered by the user, or the user enters any other key sequence that the user subsequently binds to the specified action name, `name`, in their configuration file. The `data` argument can be a pointer to anything that the application wants to have passed to the action function, `fn`, whenever that function is invoked.

The action function, `fn`, should be declared using the `GL_ACTION_FN()` macro, which is defined in `<libtecla.h>`.

```c
#define GL_ACTION_FN(fn) GlAfterAction (fn)(GetLine *gl,
    void *data, int count, size_t curpos,
    const char *line)
```

The `gl` and `data` arguments are those that were previously passed to `gl_register_action()` when the action function was registered. The `count` argument is a numeric argument which the user has the option of entering using the digit-argument action, before invoking the action. If the user does not enter a number, then the `count` argument is set to 1. Nominally this argument is interpreted as a repeat count, meaning that the action should be repeated that many times. In practice however, for some actions a repeat count makes little sense. In such cases, actions can either simply ignore the `count` argument, or use its value for a different purpose.

A copy of the current input line is passed in the read-only `line` argument. The current cursor position within this string is given by the index contained in the `curpos` argument. Note that direct manipulation of the input line and the cursor position is not permitted because the rules dictated by various modes (such as `vi` mode versus `emacs` mode, `no-echo` mode, and `insert` mode versus `overstrike` mode) make it too complex for an application writer to write a conforming editing action, as well as constrain future changes to the internals of `gl_get_line()`. A potential solution to this dilemma would be to allow the action function to edit the line using the existing editing actions. This is currently under consideration.
If the action function wishes to write text to the terminal without this getting mixed up with the displayed text of the input line, or read from the terminal without having to handle raw terminal I/O, then before doing either of these operations, it must temporarily suspend line editing by calling the `gl_normal_io()` function. This function flushes any pending output to the terminal, moves the cursor to the start of the line that follows the last terminal line of the input line, then restores the terminal to a state that is suitable for use with the C `stdio` facilities. The latter includes such things as restoring the normal mapping of \n to \r\n, and, when in server mode, restoring the normal blocking form of terminal I/O. Having called this function, the action function can read from and write to the terminal without the fear of creating a mess. It is not necessary for the action function to restore the original editing environment before it returns. This is done automatically by `gl_get_line()` after the action function returns. The following is a simple example of an action function which writes the sentence “Hello world” on a new terminal line after the line being edited. When this function returns, the input line is redrawn on the line that follows the “Hello world” line, and line editing resumes.

```c
static GL_ACTION_FN(say_hello_fn) {
    if(gl_normal_io(gl)) /* Temporarily suspend editing */
        return GLA_ABORT;
    printf("Hello world\n");
    return GLA_CONTINUE;
}
```

Action functions must return one of the following values, to tell `gl_get_line()` how to proceed.

- **GLA_ABORT** Cause `gl_get_line()` to return NULL.
- **GLA_RETURN** Cause `gl_get_line()` to return the completed input line
- **GLA_CONTINUE** Resume command-line editing.

Note that the `name` argument of `gl_register_action()` specifies the name by which a user can refer to the action in their configuration file. This allows them to re-bind the action to an alternate key-sequence. In order for this to work, it is necessary to call `gl_register_action()` between calling `new_GetLine()` and the first call to `gl_get_line()`.

History Files To save the contents of the history buffer before quitting your application and subsequently restore them when you next start the application, the `gl_save_history()` and `gl_load_history()` functions are provided.

The `filename` argument specifies the name to give the history file when saving, or the name of an existing history file, when loading. This may contain home directory and environment variable expressions, such as `~/.myapp_history` or `$HOME/.myapp_history`. 

```c

```
Along with each history line, additional information about it, such as its nesting level and when it was entered by the user, is recorded as a comment preceding the line in the history file. Writing this as a comment allows the history file to double as a command file, just in case you wish to replay a whole session using it. Since comment prefixes differ in different languages, the comment argument is provided for specifying the comment prefix. For example, if your application were a UNIX shell, such as the Bourne shell, you would specify “#” here. Whatever you choose for the comment character, you must specify the same prefix to gl_load_history() that you used when you called gl_save_history() to write the history file.

The max_lines argument must be either -1 to specify that all lines in the history list be saved, or a positive number specifying a ceiling on how many of the most recent lines should be saved.

Both functions return non-zero on error, after writing an error message to stderr. Note that gl_load_history() does not consider the non-existence of a file to be an error.

Multiple History Lists

If your application uses a single GetLine object for entering many different types of input lines, you might want gl_get_line() to distinguish the different types of lines in the history list, and only recall lines that match the current type of line. To support this requirement, gl_get_line() marks lines being recorded in the history list with an integer identifier chosen by the application. Initially this identifier is set to 0 by new_GetLine(), but it can be changed subsequently by calling gl_group_history().

The integer identifier ID can be any number chosen by the application, but note that gl_save_history() and gl_load_history() preserve the association between identifiers and historical input lines between program invocations, so you should choose fixed identifiers for the different types of input line used by your application.

Whenever gl_get_line() appends a new input line to the history list, the current history identifier is recorded with it, and when it is asked to recall a historical input line, it only recalls lines that are marked with the current identifier.

Displaying History

The history list can be displayed by calling gl_show_history(). This function displays the current contents of the history list to the stdout output stream fp. If the max_lines argument is greater than or equal to zero, then no more than this number of the most recent lines will be displayed. If the all_groups argument is non-zero, lines from all history groups are displayed. Otherwise only those of the currently selected history group are displayed. The format string argument, fmt, determines how the line is displayed. This can contain arbitrary characters which are written verbatim, interleaved with any of the following format directives:

%D  The date on which the line was originally entered, formatted like 2001-11-20.
%T  The time of day when the line was entered, formatted like 23:59:59.
%N  The sequential entry number of the line in the history buffer.
%G  The number of the history group which the line belongs to.

%%  A literal % character.

%H  The history line itself.

Thus a format string like "%D %T %H0" would output something like:
2001-11-20 10:23:34 Hello world

Note the inclusion of an explicit newline character in the format string.

Looking Up History

The gl_lookup_history() function allows the calling application to look up lines in the history list.

The id argument indicates which line to look up, where the first line that was entered in the history list after new_GetLine() was called is denoted by 0, and subsequently entered lines are denoted with successively higher numbers. Note that the range of lines currently preserved in the history list can be queried by calling the gl_range_of_history() function. If the requested line is in the history list, the details of the line are recorded in the variable pointed to by the hline argument, and 1 is returned. Otherwise 0 is returned, and the variable pointed to by hline is left unchanged.

Beware that the string returned in hline->line is part of the history buffer, so it must not be modified by the caller, and will be recycled on the next call to any function that takes gl as its argument. Therefore you should make a private copy of this string if you need to keep it.

Manual History Archival

By default, whenever a line is entered by the user, it is automatically appended to the history list, just before gl_get_line() returns the line to the caller. This is convenient for the majority of applications, but there are also applications that need finer-grained control over what gets added to the history list. In such cases, the automatic addition of entered lines to the history list can be turned off by calling the gl_automatic_history() function.

If this function is called with its enable argument set to 0, gl_get_line() will not automatically archive subsequently entered lines. Automatic archiving can be reenabled at a later time by calling this function again, with its enable argument set to 1. While automatic history archiving is disabled, the calling application can use the gl_append_history() to append lines to the history list as needed.

The line argument specifies the line to be added to the history list. This must be a normal \0 terminated string. If this string contains any newline characters, the line that gets archived in the history list will be terminated by the first of these. Otherwise it will be terminated by the \0 terminator. If the line is longer than the maximum input line length that was specified when new_GetLine() was called, it will be truncated to the actual gl_get_line() line length when the line is recalled.
If successful, `gl_append_history()` returns 0. Otherwise it returns non-zero and sets `errno` to one of the following values.

- **EINVAL**: One of the arguments passed to `gl_append_history()` was NULL.
- **ENOMEM**: The specified line was longer than the allocated size of the history buffer (as specified when `new_GetLine()` was called), so it could not be archived.

A textual description of the error can optionally be obtained by calling `gl_error_message()`. Note that after such an error, the history list remains in a valid state to receive new history lines, so there is little harm in simply ignoring the return status of `gl_append_history()`.

If you wish to change the size of the history buffer that was originally specified in the call to `new_GetLine()`, you can do so with the `gl_resize_history()` function.

The `histlen` argument specifies the new size in bytes, and if you specify this as 0, the buffer will be deleted.

As mentioned in the discussion of `new_GetLine()`, the number of lines that can be stored in the history buffer, depends on the lengths of the individual lines. For example, a 1000 byte buffer could equally store 10 lines of average length 100 bytes, or 20 lines of average length 50 bytes. Although the buffer is never expanded when new lines are added, a list of pointers into the buffer does get expanded when needed to accommodate the number of lines currently stored in the buffer. To place an upper limit on the number of lines in the buffer, and thus a ceiling on the amount of memory used in this list, you can call the `gl_limit_history()` function.

The `max_lines` should either be a positive number >= 0, specifying an upper limit on the number of lines in the buffer, or be -1 to cancel any previously specified limit. When a limit is in effect, only the `max_lines` most recently appended lines are kept in the buffer. Older lines are discarded.

To discard lines from the history buffer, use the `gl_clear_history()` function.

The `all_groups` argument tells the function whether to delete just the lines associated with the current history group (see `gl_group_history()`) or all historical lines in the buffer.

The `gl_toggle_history()` function allows you to toggle history on and off without losing the current contents of the history list.

Setting the `enable` argument to 0 turns off the history mechanism, and setting it to 1 turns it back on. When history is turned off, no new lines will be added to the history list, and history lookup key-bindings will act as though there is nothing in the history buffer.

The configured state of the history list can be queried with the `gl_history_state()` function. On return, the status information is recorded in the variable pointed to by the `state` argument.
The `gl_range_of_history()` function returns the number and range of lines in the history list. The return values are recorded in the variable pointed to by the range argument. If the `nlines` member of this structure is greater than zero, then the oldest and newest members report the range of lines in the list, and `newest=oldest+nlines-1`. Otherwise they are both zero.

The `gl_size_of_history()` function returns the total size of the history buffer and the amount of the buffer that is currently occupied.

On return, the size information is recorded in the variable pointed to by the `size` argument.

The `new_GetLine()` constructor function assumes that input is to be read from `stdin` and output written to `stdout`. The following function allows you to switch to different input and output streams.

The `gl` argument is the object that was returned by `new_GetLine()`. The `input_fp` argument specifies the stream to read from, and `output_fp` specifies the stream to be written to. Only if both of these refer to a terminal, will interactive terminal input be enabled. Otherwise `gl_get_line()` will simply call `fgets()` to read command input. If both streams refer to a terminal, then they must refer to the same terminal, and the type of this terminal must be specified with the `term` argument. The value of the `term` argument is looked up in the terminal information database (`terminfo` or `termcap`), in order to determine which special control sequences are needed to control various aspects of the terminal. `new_GetLine()` for example, passes the return value of `getenv("TERM")` in this argument. Note that if one or both of `input_fp` and `output_fp` do not refer to a terminal, then it is legal to pass `NULL` instead of a terminal type.

Note that if you want to pass file descriptors to `gl_change_terminal()`, you can do this by creating `stdio` stream wrappers using the POSIX `fdopen(3C)` function.

By default, `gl_get_line()` does not return until either a complete input line has been entered by the user, or an error occurs. In programs that need to watch for I/O from other sources than the terminal, there are two options.

- Use the functions described in the `gl_io_mode(3TECLA)` manual page to switch `gl_get_line()` into non-blocking server mode. In this mode, `gl_get_line()` becomes a non-blocking, incremental line-editing function that can safely be called from an external event loop. Although this is a very versatile method, it involves taking on some responsibilities that are normally performed behind the scenes by `gl_get_line()`.

- While `gl_get_line()` is waiting for keyboard input from the user, you can ask it to also watch for activity on arbitrary file descriptors, such as network sockets or pipes, and have it call functions of your choosing when activity is seen. This works on any system that has the select system call, which is most, if not all flavors of UNIX.
Registering a file descriptor to be watched by \texttt{gl\_get\_line()} involves calling the \texttt{gl\_watch\_fd()} function. If this returns non-zero, then it means that either your arguments are invalid, or that this facility is not supported on the host system.

The \texttt{fd} argument is the file descriptor to be watched. The event argument specifies what type of activity is of interest, chosen from the following enumerated values:

- **GLFD READ** Watch for the arrival of data to be read.
- **GLFD WRITE** Watch for the ability to write to the file descriptor without blocking.
- **GLFD URGENT** Watch for the arrival of urgent out-of-band data on the file descriptor.

The \texttt{callback} argument is the function to call when the selected activity is seen. It should be defined with the following macro, which is defined in \texttt{libtecla.h}.

\begin{verbatim}
#define GL_FD_EVENT_FN(fn) GlFdStatus (fn)(GetLine *gl, 
    void *data, int fd, GlFdEvent event)
\end{verbatim}

The data argument of the \texttt{gl\_watch\_fd()} function is passed to the callback function for its own use, and can point to anything you like, including NULL. The file descriptor and the event argument are also passed to the callback function, and this potentially allows the same callback function to be registered to more than one type of event and/or more than one file descriptor.

The return value of the callback function should be one of the following values.

- **GLFD_ABORT** Tell \texttt{gl\_get\_line()} to abort. When this happens, \texttt{gl\_get\_line()} returns NULL, and a following call to \texttt{gl\_return\_status()} will return GLR_FDABORT. Note that if the application needs \texttt{errno} always to have a meaningful value when \texttt{gl\_get\_line()} returns NULL, the callback function should set \texttt{errno} appropriately.

- **GLFD_REFRESH** Redraw the input line then continue waiting for input. Return this if your callback wrote to the terminal.

- **GLFD_CONTINUE** Continue to wait for input, without redrawing the line.

Note that before calling the callback, \texttt{gl\_get\_line()} blocks most signals and leaves its own signal handlers installed, so if you need to catch a particular signal you will need to both temporarily install your own signal handler, and unblock the signal. Be sure to re-block the signal (if it was originally blocked) and reinstate the original signal handler, if any, before returning.

Your callback should not try to read from the terminal, which is left in raw mode as far as input is concerned. You can write to the terminal as usual, since features like conversion of newline to carriage-return/linefeed are re-enabled while the callback is running. If your callback function does write to the terminal, be sure to output a newline first, and when your callback returns, tell \texttt{gl\_get\_line()} that the input line needs to be redrawn, by returning the GLFD_REFRESH status code.
To remove a callback function that you previously registered for a given file descriptor and event, simply call `gl_watch_fd()` with the same `fd` and `event` arguments, but with a `callback` argument of 0. The `data` argument is ignored in this case.

The `gl_inactivity_timeout()` function can be used to set or cancel an inactivity timeout. Inactivity in this case refers both to keyboard input, and to I/O on any file descriptors registered by prior and subsequent calls to `gl_watch_fd()`.

The timeout is specified in the form of an integral number of seconds and an integral number of nanoseconds, specified by the `sec` and `nsec` arguments, respectively. Subsequently, whenever no activity is seen for this time period, the function specified by the `callback` argument is called. The `data` argument of `gl_inactivity_timeout()` is passed to this callback function whenever it is invoked, and can thus be used to pass arbitrary application-specific information to the callback. The following macro is provided in `<libtecla.h>` for applications to use to declare and prototype timeout callback functions.

```
#define GL_TIMEOUT_FN(fn) GlAfterTimeout (fn)(GetLine *gl, void *data)
```

On returning, the application’s callback is expected to return one of the following enumerators to tell `gl_get_line()` how to procede after the timeout has been handled by the callback.

- **GLTO_ABORT**
  - Tell `gl_get_line()` to abort. When this happens, `gl_get_line()` will return `NULL`, and a following call to `gl_return_status()` will return `GLR_TIMEOUT`. Note that if the application needs `errno` already to have a meaningful value when `gl_get_line()` returns `NULL`, the callback function should set `errno` appropriately.

- **GLTO_REFRESH**
  - Redraw the input line, then continue waiting for input. You should return this value if your callback wrote to the terminal.

- **GLTO_CONTINUE**
  - In normal blocking-I/O mode, continue to wait for input, without redrawing the user’s input line. In non-blocking server I/O mode (see `gl_io_mode(3TECLA)`), `gl_get_line()` acts as though I/O blocked. This means that `gl_get_line()` will immediately return `NULL`, and a following call to `gl_return_status()` will return `GLR_BLOCKED`.

Note that before calling the callback, `gl_get_line()` blocks most signals and leaves its own signal handlers installed, so if you need to catch a particular signal you will need to both temporarily install your own signal handler and unblock the signal. Be sure to re-block the signal (if it was originally blocked) and reinstate the original signal handler, if any, before returning.

Your callback should not try to read from the terminal, which is left in raw mode as far as input is concerned. You can however write to the terminal as usual, since features like conversion of newline to carriage-return/linefeed are re-enabled while the callback is running.
If your callback function does write to the terminal, be sure to output a newline first, and when your callback returns, tell gl_get_line() that the input line needs to be redrawn, by returning the GLTO_REFRESH status code.

Finally, note that although the timeout arguments include a nanosecond component, few computer clocks presently have resolutions that are finer than a few milliseconds, so asking for less than a few milliseconds is equivalent to requesting zero seconds on many systems. If this would be a problem, you should base your timeout selection on the actual resolution of the host clock (for example, by calling sysconf(_SC_CLK_TCK)).

To turn off timeouts, simply call gl_inactivity_timeout() with a callback argument of 0. The data argument is ignored in this case.

Signal Handling Defaults

By default, the gl_get_line() function intercepts a number of signals. This is particularly important for signals that would by default terminate the process, since the terminal needs to be restored to a usable state before this happens. This section describes the signals that are trapped by default and how gl_get_line() responds to them. Changing these defaults is the topic of the following section.

When the following subset of signals are caught, gl_get_line() first restores the terminal settings and signal handling to how they were before gl_get_line() was called, resends the signal to allow the calling application's signal handlers to handle it, then, if the process still exists, returns NULL and sets errno as specified below.

SIGINT  This signal is generated both by the keyboard interrupt key (usually ^C), and the
         keyboard break key. The errno value is EINTR.

SIGHUP  This signal is generated when the controlling terminal exits. The errno value is
         ENOTTY.

SIGPIPE  This signal is generated when a program attempts to write to a pipe whose
         remote end is not being read by any process. This can happen for example if you
         have called gl_change_terminal() to redirect output to a pipe hidden under a
         pseudo terminal. The errno value is EPIPE.

SIGQUIT  This signal is generated by the keyboard quit key (usually ^\). The errno value is
         EINTR.

SIGABRT  This signal is generated by the standard C, abort function. By default it both
         terminates the process and generates a core dump. The errno value is EINTR.

SIGTERM  This is the default signal that the UNIX kill command sends to processes. The
         errno value is EINTR.

Note that in the case of all of the above signals, POSIX mandates that by default the process is
terminated, with the addition of a core dump in the case of the SIGQUIT signal. In other words,
if the calling application does not override the default handler by supplying its own signal handler, receipt of the corresponding signal will terminate the application before gl_get_line() returns.

If gl_get_line() aborts with errno set to EINTR, you can find out what signal caused it to abort, by calling the gl_last_signal() function. This returns the numeric code (for example, SIGINT) of the last signal that was received during the most recent call to gl_get_line(), or -1 if no signals were received.

On systems that support it, when a SIGWINCH (window change) signal is received, gl_get_line() queries the terminal to find out its new size, redraws the current input line to accommodate the new size, then returns to waiting for keyboard input from the user. Unlike other signals, this signal is not resent to the application.

Finally, the following signals cause gl_get_line() to first restore the terminal and signal environment to that which prevailed before gl_get_line() was called, then resend the signal to the application. If the process still exists after the signal has been delivered, then gl_get_line() then re-establishes its own signal handlers, switches the terminal back to raw mode, redisplay the input line, and goes back to awaiting terminal input from the user.

SIGCONT This signal is generated when a suspended process is resumed.
SIGPOLL On SVR4 systems, this signal notifies the process of an asynchronous I/O event. Note that under 4.3BSD, SIGIO and SIGPOLL are the same. On other systems, SIGIO is ignored by default, so gl_get_line() does not trap it by default.
SIGPWR This signal is generated when a power failure occurs (presumably when the system is on a UPS).
SIGALRM This signal is generated when a timer expires.
SIGUSR1 An application specific signal.
SIGUSR2 Another application specific signal.
SIGTALRM This signal is generated when a virtual timer expires. See setitimer(2).
SIGXCPU This signal is generated when a process exceeds its soft CPU time limit.
SIGXFSZ This signal is generated when a process exceeds its soft file-size limit.
SIGTSTP This signal is generated by the terminal suspend key, which is usually ^Z, or the delayed terminal suspend key, which is usually ^Y.
SIGTTIN This signal is generated if the program attempts to read from the terminal while the program is running in the background.
SIGTTOU This signal is generated if the program attempts to write to the terminal while the program is running in the background.
Obviously not all of the above signals are supported on all systems, so code to support them is conditionally compiled into the tecla library.

Note that if SIGKILL or SIGPOLL, which by definition cannot be caught, or any of the hardware generated exception signals, such as SIGSEGV, SIGBUS, and SIGFPE, are received and unhandled while gl_get_line() has the terminal in raw mode, the program will be terminated without the terminal having been restored to a usable state. In practice, job-control shells usually reset the terminal settings when a process relinquishes the controlling terminal, so this is only a problem with older shells.

The previous section listed the signals that gl_get_line() traps by default, and described how it responds to them. This section describes how to both add and remove signals from the list of trapped signals, and how to specify how gl_get_line() should respond to a given signal.

If you do not need gl_get_line() to do anything in response to a signal that it normally traps, you can tell to gl_get_line() to ignore that signal by calling gl_ignore_signal().

The signo argument is the number of the signal (for example, SIGINT) that you want to have ignored. If the specified signal is not currently one of those being trapped, this function does nothing.

The gl_trap_signal() function allows you to either add a new signal to the list that gl_get_line() traps or modify how it responds to a signal that it already traps.

The signo argument is the number of the signal that you want to have trapped. The flags argument is a set of flags that determine the environment in which the application’s signal handler is invoked. The after argument tells gl_get_line() what to do after the application’s signal handler returns. The errno_value tells gl_get_line() what to set errno to if told to abort.

The flags argument is a bitwise OR of zero or more of the following enumerators:

- **GLS_RESTORE_SIG**  
  Restore the caller’s signal environment while handling the signal.

- **GLS_RESTORE_TTY**  
  Restore the caller’s terminal settings while handling the signal.

- **GLS_RESTORE_LINE**  
  Move the cursor to the start of the line following the input line before invoking the application’s signal handler.

- **GLS_REDRAW_LINE**  
  Redraw the input line when the application’s signal handler returns.

- **GLS_UNBLOCK_SIG**  
  Normally, if the calling program has a signal blocked (see sigprocmask(2)), gl_get_line() does not trap that signal. This flag tells gl_get_line() to trap the signal and unblock it for the duration of the call to gl_get_line().
GLS_DONT_FORWARD If this flag is included, the signal will not be forwarded to the signal handler of the calling program.

Two commonly useful flag combinations are also enumerated as follows:

<table>
<thead>
<tr>
<th>GLS_RESTORE_ENV</th>
<th>GLS_RESTORE_SIG</th>
<th>GLS_RESTORE_TTY</th>
<th>GLS_REDRAW_LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLS_SUSPEND_INPUT</td>
<td>GLS_RESTORE_ENV</td>
<td>GLS_RESTORE_LINE</td>
<td></td>
</tr>
</tbody>
</table>

If your signal handler, or the default system signal handler for this signal, if you have not overridden it, never either writes to the terminal, nor suspends or terminates the calling program, then you can safely set the flags argument to 0.

- The cursor does not get left in the middle of the input line.
- So that the user can type in input and have it echoed.
- So that you do not need to end each output line with \n, instead of just \n.

The GL_RESTORE_ENV combination is the same as GL_SUSPEND_INPUT, except that it does not move the cursor. If your signal handler does not read or write anything to the terminal, the user will not see any visible indication that a signal was caught. This can be useful if you have a signal handler that only occasionally writes to the terminal, where using GL_SUSPEND_LINE would cause the input line to be unnecessarily duplicated when nothing had been written to the terminal. Such a signal handler, when it does write to the terminal, should be sure to start a new line at the start of its first write, by writing a new line before returning. If the signal arrives while the user is entering a line that only occupies a signal terminal line, or if the cursor is on the last terminal line of a longer input line, this will have the same effect as GL_SUSPEND_INPUT. Otherwise it will start writing on a line that already contains part of the displayed input line. This does not do any harm, but it looks a bit ugly, which is why the GL_SUSPEND_INPUT combination is better if you know that you are always going to be writing to the terminal.

The after argument, which determines what gl_get_line() does after the application’s signal handler returns (if it returns), can take any one of the following values:

| GLS_RETURN | Return the completed input line, just as though the user had pressed the return key. |
| GLS_ABORT | Cause gl_get_line() to abort. When this happens, gl_get_line() returns NULL, and a following call to gl_return_status() will return GLR_SIGNAL. Note that if the application needs errno always to have a meaningful value when gl_get_line() returns NULL, the callback function should set errno appropriately. |
| GLS_CONTINUE | Resume command line editing. |

The errno_value argument is intended to be combined with the GLS_ABORT option, telling gl_get_line() what to set the standard errno variable to before returning NULL to the calling
program. It can also, however, be used with the GL_RETURN option, in case you want to have a way to distinguish between an input line that was entered using the return key, and one that was entered by the receipt of a signal.

Reliable Signal Handling

Signal handling is surprisingly hard to do reliably without race conditions. In gl_get_line() a lot of care has been taken to allow applications to perform reliable signal handling around gl_get_line(). This section explains how to make use of this.

As an example of the problems that can arise if the application is not written correctly, imagine that one's application has a SIGINT signal handler that sets a global flag. Now suppose that the application tests this flag just before invoking gl_get_line(). If a SIGINT signal happens to be received in the small window of time between the statement that tests the value of this flag, and the statement that calls gl_get_line(), then gl_get_line() will not see the signal, and will not be interrupted. As a result, the application will not be able to respond to the signal until the user gets around to finishing entering the input line and gl_get_line() returns. Depending on the application, this might or might not be a disaster, but at the very least it would puzzle the user.

The way to avoid such problems is to do the following.

1. If needed, use the gl_trap_signal() function to configure gl_get_line() to abort when important signals are caught.

2. Configure gl_get_line() such that if any of the signals that it catches are blocked when gl_get_line() is called, they will be unblocked automatically during times when gl_get_line() is waiting for I/O. This can be done either on a per signal basis, by calling the gl_trap_signal() function, and specifying the GLS_UNBLOCK attribute of the signal, or globally by calling the gl_catch_blocked() function. This function simply adds the GLS_UNBLOCK attribute to all of the signals that it is currently configured to trap.

3. Just before calling gl_get_line(), block delivery of all of the signals that gl_get_line() is configured to trap. This can be done using the POSIX sigprocmask function in conjunction with the gl_list_signals() function. This function returns the set of signals that it is currently configured to catch in the set argument, which is in the form required by sigprocmask().

4. In the example, one would now test the global flag that the signal handler sets, knowing that there is now no danger of this flag being set again until gl_get_line() unblocks its signals while performing I/O.

5. Eventually gl_get_line() returns, either because a signal was caught, an error occurred, or the user finished entering their input line.

6. Now one would check the global signal flag again, and if it is set, respond to it, and zero the flag.

7. Use sigprocmask() to unblock the signals that were blocked in step 3.
The same technique can be used around certain POSIX signal-aware functions, such as `sigsetjmp(3C)` and `sigsuspend(2)`, and in particular, the former of these two functions can be used in conjunction with `siglongjmp(3C)` to implement race-condition free signal handling around other long-running system calls. The `gl_get_line()` function manages to reliably trap signals around calls to functions like `read(2)` and `select(3C)` without race conditions.

The `gl_get_line()` function first uses the POSIX `sigprocmask()` function to block the delivery of all of the signals that it is currently configured to catch. This is redundant if the application has already blocked them, but it does no harm. It undoes this step just before returning.

Whenever `gl_get_line()` needs to call `read` or `select` to wait for input from the user, it first calls the POSIX `sigsetjmp()` function, being sure to specify a non-zero value for its `savemask` argument.

If `sigsetjmp()` returns zero, `gl_get_line()` then does the following.

1. It uses the POSIX `sigaction(2)` function to register a temporary signal handler to all of the signals that it is configured to catch. This signal handler does two things.
   a. It records the number of the signal that was received in a file-scope variable.
   b. It then calls the POSIX `siglongjmp()` function using the buffer that was passed to `sigsetjmp()` for its first argument and a non-zero value for its second argument.

   When this signal handler is registered, the `sa_mask` member of the `struct sigaction act` argument of the call to `sigaction()` is configured to contain all of the signals that `gl_get_line()` is catching. This ensures that only one signal will be caught at once by our signal handler, which in turn ensures that multiple instances of our signal handler do not tread on each other's toes.

2. Now that the signal handler has been set up, `gl_get_line()` unblocks all of the signals that it is configured to catch.

3. It then calls the `read()` or `select()` function to wait for keyboard input.

4. If this function returns (that is, no signal is received), `gl_get_line()` blocks delivery of the signals of interest again.

5. It then reinstates the signal handlers that were displaced by the one that was just installed.

Alternatively, if `sigsetjmp()` returns non-zero, this means that one of the signals being trapped was caught while the above steps were executing. When this happens, `gl_get_line()` does the following.

First, note that when a call to `siglongjmp()` causes `sigsetjmp()` to return, provided that the `savemask` argument of `sigsetjmp()` was non-zero, the signal process mask is restored to how it was when `sigsetjmp()` was called. This is the important difference between `sigsetjmp()`
and the older problematic `setjmp(3C)`, and is the essential ingredient that makes it possible to avoid signal handling race conditions. Because of this we are guaranteed that all of the signals that we blocked before calling `sigsetjmp()` are blocked again as soon as any signal is caught. The following statements, which are then executed, are thus guaranteed to be executed without any further signals being caught.

1. If so instructed by the `gl_get_line()` configuration attributes of the signal that was caught, `gl_get_line()` restores the terminal attributes to the state that they had when `gl_get_line()` was called. This is particularly important for signals that suspend or terminate the process, since otherwise the terminal would be left in an unusable state.

2. It then reinstates the application’s signal handlers.

3. Then it uses the C standard-library `raise(3C)` function to re-send the application the signal that was caught.

4. Next it unblocks delivery of the signal that we just sent. This results in the signal that was just sent by `raise()` being caught by the application’s original signal handler, which can now handle it as it sees fit.

5. If the signal handler returns (that is, it does not terminate the process), `gl_get_line()` blocks delivery of the above signal again.

6. It then undoes any actions performed in the first of the above steps and redisplays the line, if the signal configuration calls for this.

7. `gl_get_line()` then either resumes trying to read a character, or aborts, depending on the configuration of the signal that was caught.

What the above steps do in essence is to take asynchronously delivered signals and handle them synchronously, one at a time, at a point in the code where `gl_get_line()` has complete control over its environment.

### The Terminal Size

On most systems the combination of the `TIOCGWINSZ` ioctl and the SIGWINCH signal is used to maintain an accurate idea of the terminal size. The terminal size is newly queried every time that `gl_get_line()` is called and whenever a SIGWINCH signal is received.

On the few systems where this mechanism is not available, at startup `new_GetLine()` first looks for the LINES and COLUMNS environment variables. If these are not found, or they contain unusable values, then if a terminal information database like `terminfo` or `termcap` is available, the default size of the terminal is looked up in this database. If this too fails to provide the terminal size, a default size of 80 columns by 24 lines is used.

Even on systems that do support ioctl(TIOCGWINSZ), if the terminal is on the other end of a serial line, the terminal driver generally has no way of detecting when a resize occurs or of querying what the current size is. In such cases no SIGWINCH is sent to the process, and the dimensions returned by ioctl(TIOCGWINSZ) are not correct. The only way to handle such
instances is to provide a way for the user to enter a command that tells the remote system what the new size is. This command would then call the `gl_set_term_size()` function to tell `gl_get_line()` about the change in size.

The `ncolumn` and `nline` arguments are used to specify the new dimensions of the terminal, and must not be less than 1. On systems that do support `ioctl(TIOCGWINSZ)`, this function first calls `ioctl(TIOCSWINSZ)` to tell the terminal driver about the change in size. In non-blocking server-I/O mode, if a line is currently being input, the input line is then redrawn to accommodate the changed size. Finally the new values are recorded in `gl` for future use by `gl_get_line()`.

The `gl_terminal_size()` function allows you to query the current size of the terminal, and install an alternate fallback size for cases where the size is not available. Beware that the terminal size will not be available if reading from a pipe or a file, so the default values can be important even on systems that do support ways of finding out the terminal size.

This function first updates `gl_get_line()`’s fallback terminal dimensions, then records its findings in the return value.

The `def_ncolumn` and `def_nline` arguments specify the default number of terminal columns and lines to use if the terminal size cannot be determined by `ioctl(TIOCGWINSZ)` or environment variables.

Hiding What You Type

When entering sensitive information, such as passwords, it is best not to have the text that you are entering echoed on the terminal. Furthermore, such text should not be recorded in the history list, since somebody finding your terminal unattended could then recall it, or somebody snooping through your directories could see it in your history file. With this in mind, the `gl_echo_mode()` function allows you to toggle on and off the display and archival of any text that is subsequently entered in calls to `gl_get_line()`.

The `enable` argument specifies whether entered text should be visible or not. If it is 0, then subsequently entered lines will not be visible on the terminal, and will not be recorded in the history list. If it is 1, then subsequent input lines will be displayed as they are entered, and provided that history has not been turned off with a call to `gl_toggle_history()`, then they will also be archived in the history list. Finally, if the enable argument is -1, then the echoing mode is left unchanged, which allows you to non-destructively query the current setting through the return value. In all cases, the return value of the function is 0 if echoing was disabled before the function was called, and 1 if it was enabled.

When echoing is turned off, note that although tab completion will invisibly complete your prefix as far as possible, ambiguous completions will not be displayed.
Single Character Queries

Using `gl_get_line()` to query the user for a single character reply, is inconvenient for the user, since they must hit the enter or return key before the character that they typed is returned to the program. Thus the `gl_query_char()` function has been provided for single character queries like this.

This function displays the specified prompt at the start of a new line, and waits for the user to type a character. When the user types a character, `gl_query_char()` displays it to the right of the prompt, starts a newline, then returns the character to the calling program. The return value of the function is the character that was typed. If the read had to be aborted for some reason, EOF is returned instead. In the latter case, the application can call the previously documented `gl_return_status()`, to find out what went wrong. This could, for example, have been the reception of a signal, or the optional inactivity timer going off.

If the user simply hits enter, the value of the `defchar` argument is substituted. This means that when the user hits either new line or return, the character specified in `defchar`, is displayed after the prompt, as though the user had typed it, as well as being returned to the calling application. If such a replacement is not important, simply pass `\n` as the value of `defchar`.

If the entered character is an unprintable character, it is displayed symbolically. For example, control-A is displayed as `^A`, and characters beyond 127 are displayed in octal, preceded by a backslash.

As with `gl_get_line()`, echoing of the entered character can be disabled using the `gl_echo_mode()` function.

If the calling process is suspended while waiting for the user to type their response, the cursor is moved to the line following the prompt line, then when the process resumes, the prompt is redisplayed, and `gl_query_char()` resumes waiting for the user to type a character.

Note that in non-blocking server mode, if an incomplete input line is in the process of being read when `gl_query_char()` is called, the partial input line is discarded, and erased from the terminal, before the new prompt is displayed. The next call to `gl_get_line()` will thus start editing a new line.

Reading Raw Characters

Whereas the `gl_query_char()` function visibly prompts the user for a character, and displays what they typed, the `gl_read_char()` function reads a signal character from the user, without writing anything to the terminal, or perturbing any incompletely entered input line. This means that it can be called not only from between calls to `gl_get_line()`, but also from callback functions that the application has registered to be called by `gl_get_line()`.

On success, the return value of `gl_read_char()` is the character that was read. On failure, EOF is returned, and the `gl_return_status()` function can be called to find out what went wrong. Possibilities include the optional inactivity timer going off, the receipt of a signal that is configured to abort `gl_get_line()`, or terminal I/O blocking, when in non-blocking server-I/O mode.
Beware that certain keyboard keys, such as function keys, and cursor keys, usually generate at least three characters each, so a single call to `gl_read_char()` will not be enough to identify such keystrokes.

### Clearing the Terminal

The calling program can clear the terminal by calling `gl_erase_terminal()`. In non-blocking server-I/O mode, this function also arranges for the current input line to be redrawn from scratch when `gl_get_line()` is next called.

### Displaying Text Dynamically

Between calls to `gl_get_line()`, the `gl_display_text()` function provides a convenient way to display paragraphs of text, left-justified and split over one or more terminal lines according to the constraints of the current width of the terminal. Examples of the use of this function may be found in the demo programs, where it is used to display introductions. In those examples the advanced use of optional prefixes, suffixes and filled lines to draw a box around the text is also illustrated.

If `gl` is not currently connected to a terminal, for example if the output of a program that uses `gl_get_line()` is being piped to another program or redirected to a file, then the value of the `def_width` parameter is used as the terminal width.

The `indentation` argument specifies the number of characters to use to indent each line of output. The `fill_char` argument specifies the character that will be used to perform this indentation.

The `prefix` argument can be either `NULL` or a string to place at the beginning of each new line (after any indentation). Similarly, the `suffix` argument can be either `NULL` or a string to place at the end of each line. The suffix is placed flush against the right edge of the terminal, and any space between its first character and the last word on that line is filled with the character specified by the `fill_char` argument. Normally the fill-character is a space.

The `start` argument tells `gl_display_text()` how many characters have already been written to the current terminal line, and thus tells it the starting column index of the cursor. Since the return value of `gl_display_text()` is the ending column index of the cursor, by passing the return value of one call to the start argument of the next call, a paragraph that is broken between more than one string can be composed by calling `gl_display_text()` for each successive portion of the paragraph. Note that literal newline characters are necessary at the end of each paragraph to force a new line to be started.

On error, `gl_display_text()` returns -1.

### Callback Function Facilities

Unless otherwise stated, callback functions such as tab completion callbacks and event callbacks should not call any functions in this module. The following functions, however, are designed specifically to be used by callback functions.

Calling the `gl_replace_prompt()` function from a callback tells `gl_get_line()` to display a different prompt when the callback returns. Except in non-blocking server mode, it has no effect if used between calls to `gl_get_line()`. In non-blocking server mode, when used
between two calls to `gl_get_line()` that are operating on the same input line, the current input line will be re-drawn with the new prompt on the following call to `gl_get_line()`.

Since `libtecla(3LIB)` version 1.4.0, `gl_get_line()` has been 8-bit clean. This means that all 8-bit characters that are printable in the user's current locale are now displayed verbatim and included in the returned input line. Assuming that the calling program correctly contains a call like the following,

```c
setlocale(LC_CTYPE, "")
```

then the current locale is determined by the first of the environment variables `LC_CTYPE`, `LC_ALL`, and `LANG` that is found to contain a valid locale name. If none of these variables are defined, or the program neglects to call `setlocale(3C)`, then the default C locale is used, which is US 7-bit ASCII. On most UNIX-like platforms, you can get a list of valid locales by typing the command:

```
locale -a
```

at the shell prompt. Further documentation on how the user can make use of this to enter international characters can be found in the `tecla(5)` man page.

Unfortunately neither `terminfo` nor `termcap` were designed to be reentrant, so you cannot safely use the functions of the getline module in multiple threads (you can use the separate file-expansion and word-completion modules in multiple threads, see the corresponding man pages for details). However due to the use of POSIX reentrant functions for looking up home directories, it is safe to use this module from a single thread of a multi-threaded program, provided that your other threads do not use any `termcap` or `terminfo` functions.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`cpl_complete_word(3TECLA), ef_expand_file(3TECLA), gl_io_mode(3TECLA),
libtecla(3LIB), pca_lookup_file(3TECLA), attributes(5), tecla(5)`
The `gl_get_line(3TECLA)` function supports two different I/O modes. These are selected by calling the `gl_io_mode()` function. The `mode` argument of `gl_io_mode()` specifies the new I/O mode and must be one of the following.

**Server I/O Mode**

- **GL_NORMAL_MODE**
  - Select the normal blocking-I/O mode. In this mode `gl_get_line()` does not return until either an error occurs of the user finishes entering a new line.

- **GL_SERVER_MODE**
  - Select non-blocking server I/O mode. In this mode, since non-blocking terminal I/O is used, the entry of each new input line typically requires many calls to `gl_get_line()` from an external I/O-driven event loop.

Newly created GetLine objects start in normal I/O mode, so to switch to non-blocking server mode requires an initial call to `gl_io_mode()`.

In non-blocking server I/O mode, the application is required to have an event loop that calls `gl_get_line()` whenever the terminal file descriptor can perform the type I/O that `gl_get_line()` is waiting for. To determine which type of I/O `gl_get_line()` is waiting for, the application calls the `gl_pending_io()` function. The return value is one of the following two enumerated values.

- **GLP_READ**`gl_get_line()` is waiting to write a character to the terminal.
- **GLP_WRITE**`gl_get_line()` is waiting to read a character from the keyboard.

If the application is using either the `select(3C)` or `poll(2)` function to watch for I/O on a group of file descriptors, then it should call the `gl_pending_io()` function before each call to these functions to determine which direction of I/O it should tell them to watch for, and configure their arguments accordingly. In the case of the `select()` function, this means using
the FD_SET() macro to add the terminal file descriptor either to the set of file descriptors to be watched for readability or the set to be watched for writability.

As in normal I/O mode, the return value of gl_get_line() is either a pointer to a completed input line or NULL. However, whereas in normal I/O mode a NULL return value always means that an error occurred, in non-blocking server mode, NULL is also returned when gl_get_line() cannot read or write to the terminal without blocking. Thus in non-blocking server mode, in order to determine when a NULL return value signifies that an error occurred or not, it is necessary to call the gl_return_status() function. If this function returns the enumerated value GLR_BLOCKED, gl_get_line() is waiting for I/O and no error has occurred.

When gl_get_line() returns NULL and gl_return_status() indicates that this is due to blocked terminal I/O, the application should call gl_get_line() again when the type of I/O reported by gl_pending_io() becomes possible. The prompt, start_line and start_pos arguments of gl_get_line() will be ignored on these calls. If you need to change the prompt of the line that is currently being edited, you can call the gl_replace_prompt() function between calls to gl_get_line().

A complication that is unique to non-blocking server mode is that it requires that the terminal be left in raw mode between calls to gl_get_line(). If this were not the case, the external event loop would not be able to detect individual key-presses, and the basic line editing implemented by the terminal driver would clash with the editing provided by gl_get_line(). When the terminal needs to be used for purposes other than entering a new input line with gl_get_line(), it needs to be restored to a usable state. In particular, whenever the process is suspended or terminated, the terminal must be returned to a normal state. If this is not done, then depending on the characteristics of the shell that was used to invoke the program, the user could end up with a hung terminal. To this end, the gl_normal_io() function is provided for switching the terminal back to the state that it was in when raw mode was last established.

The gl_normal_io() function first flushes any pending output to the terminal, then moves the cursor to the start of the terminal line which follows the end of the incompletely entered input line. At this point it is safe to suspend or terminate the process, and it is safe for the application to read and write to the terminal. To resume entry of the input line, the application should call the gl_raw_io() function.

The gl_normal_io() function starts a new line, redisplay the partially completed input line (if any), restores the cursor position within this line to where it was when gl_normal_io() was called, then switches back to raw, non-blocking terminal mode ready to continue entry of the input line when gl_get_line() is next called.

Note that in non-blocking server mode, if gl_get_line() is called after a call to gl_normal_io(), without an intervening call to gl_raw_io(), gl_get_line() will call gl_raw_mode() itself, and the terminal will remain in this mode when gl_get_line() returns.
In the previous section it was pointed out that in non-blocking server mode, the terminal must be restored to a sane state whenever a signal is received that either suspends or terminates the process. In normal I/O mode, this is done for you by `gl_get_line()`, but in non-blocking server mode, since the terminal is left in raw mode between calls to `gl_get_line()`, this signal handling has to be done by the application. Since there are many signals that can suspend or terminate a process, as well as other signals that are important to `gl_get_line()`, such as the SIGWINCH signal, which tells it when the terminal size has changed, the `gl_tty_signals()` function is provided for installing signal handlers for all pertinent signals.

The `gl_tty_signals()` function uses `gl_get_line()`’s internal list of signals to assign specified signal handlers to groups of signals. The arguments of this function are as follows.

- **term_handler**: This is the signal handler that is used to trap signals that by default terminate any process that receives them (for example, SIGINT or SIGTERM).
- **susp_handler**: This is the signal handler that is used to trap signals that by default suspend any process that receives them, (for example, SIGTSTP or SIGTT0U).
- **cont_handler**: This is the signal handler that is used to trap signals that are usually sent when a process resumes after being suspended (usually SIGCONT). Beware that there is nothing to stop a user from sending one of these signals at other times.
- **size_handler**: This signal handler is used to trap signals that are sent to processes when their controlling terminals are resized by the user (for example, SIGWINCH).

These arguments can all be the same, if so desired, and SIG_IGN (ignore this signal) or SIG_DFL (use the system-provided default signal handler) can be specified instead of a function where pertinent. In particular, it is rarely useful to trap SIGCONT, so the cont_handler argument will usually be SIG_DFL or SIG_IGN.

The `gl_tty_signals()` function uses the POSIX `sigaction(2)` function to install these signal handlers, and it is careful to use the `sa_mask` member of each `sigaction` structure to ensure that only one of these signals is ever delivered at a time. This guards against different instances of these signal handlers from simultaneously trying to write to common global data, such as a shared `sigsetjmp(3C)` buffer or a signal-received flag. The signal handlers installed by this function should call the `gl_handle_signal()`.

The `signo` argument tells this function which signal it is being asked to respond to, and the `gl` argument should be a pointer to the first element of an array of `ngl_get_line` objects. If your application has only one of these objects, pass its pointer as the `gl` argument and specify `ngl` as 1.

Depending on the signal that is being handled, this function does different things.
If the signal that was caught is one of those that by default terminates any process that receives it, then `gl_handle_signal()` does the following steps.

1. First it blocks the delivery of all signals that can be blocked (i.e. SIGKILL and SIGSTOP cannot be blocked).
2. Next it calls `gl_normal_io()` for each of the ngl GetLine objects. Note that this does nothing to any of the GetLine objects that are not currently in raw mode.
3. Next it sets the signal handler of the signal to its default, process-termination disposition.
4. Next it re-sends the process the signal that was caught.
5. Finally it unblocks delivery of this signal, which results in the process being terminated.

If the default disposition of the signal is to suspend the process, the same steps are executed as for process termination signals, except that when the process is later resumed, `gl_handle_signal()` continues, and does the following steps.

1. It re-blocks delivery of the signal.
2. It reinstates the signal handler of the signal to the one that was displaced when its default disposition was substituted.
3. For any of the GetLine objects that were in raw mode when `gl_handle_signal()` was called, `gl_handle_signal()` then calls `gl_raw_io()`, to resume entry of the input lines on those terminals.
4. Finally, it restores the signal process mask to how it was when `gl_handle_signal()` was called.

Note that the process is suspended or terminated using the original signal that was caught, rather than using the uncatchable SIGSTOP and SIGKILL signals. This is important, because when a process is suspended or terminated, the parent of the process may wish to use the status value returned by the wait system call to figure out which signal was responsible. In particular, most shells use this information to print a corresponding message to the terminal. Users would be rightly confused if when their process received a SIGPIPE signal, the program responded by sending itself a SIGKILL signal, and the shell then printed out the provocative statement, "Killed!".

If a signal is caught and handled when the application's event loop is waiting in `select()` or `poll()`, these functions will be aborted with `errno` set to EINTR. When this happens the event loop should call `gl_pending_io()` before calling `select()` or `poll()` again. It should then arrange for `select()` or `poll()` to wait for the type of I/O that `gl_pending_io()` reports. This is necessary because any signal handler that calls `gl_handle_signal()` will frequently change the type of I/O that `gl_get_line()` is waiting for.

If a signal arrives between the statements that configure the arguments of `select()` or `poll()` and the calls to these functions, the signal will not be seen by these functions, which will then not be aborted. If these functions are waiting for keyboard input from the user when the signal
is received, and the signal handler arranges to redraw the input line to accommodate a
terminal resize or the resumption of the process. This redisplay will be delayed until the user
presses the next key. Apart from puzzling the user, this clearly is not a serious problem.
However there is a way, albeit complicated, to completely avoid this race condition. The
following steps illustrate this.

1. Block all of the signals that `gl_get_line()` catches, by passing the signal set returned by
   `gl_list_signals()` to `sigprocmask()`.
2. Call `gl_pending_io()` and set up the arguments of `select()` or `poll()` accordingly.
3. Call `sigsetjmp()` with a non-zero `savemask` argument.
4. Initially this `sigsetjmp()` statement will return zero, indicating that control is not
   resuming there after a matching call to `siglongjmp()`.
5. Replace all of the handlers of the signals that `gl_get_line()` is configured to catch, with a
   signal handler that first records the number of the signal that was caught, in a file-scope
   variable, then calls `siglongjmp()` with a non-zero `val` argument, to return execution to the
   above `sigsetjmp()` statement. Registering these signal handlers can conveniently be done
   using the `gl_tty_signals()` function.
6. Set the file-scope variable that the above signal handler uses to record any signal that is
   caught to -1, so that we can check whether a signal was caught by seeing if it contains a
   valid signal number.
7. Now unblock the signals that were blocked in step 1. Any signal that was received by the
   process in between step 1 and now will now be delivered, and trigger our signal handler, as
   will any signal that is received until we block these signals again.
8. Now call `select()` or `poll()`.
9. When `select()` returns, again block the signals that were unblocked in step 7.
   If a signal is arrived any time during the above steps, our signal handler will be triggered
   and cause control to return to the `sigsetjmp()` statement, where this time, `sigsetjmp()`
   will return non-zero, indicating that a signal was caught. When this happens we simply
   skip the above block of statements, and continue with the following statements, which are
   executed regardless of whether or not a signal is caught. Note that when `sigsetjmp()`
   returns, regardless of why it returned, the process signal mask is returned to how it was
   when `sigsetjmp()` was called. Thus the following statements are always executed with all
   of our signals blocked.
10. Reinstate the signal handlers that were displaced in step 5.
11. Check wether a signal was caught, by checking the file-scope variable that the signal
    handler records signal numbers in.
12. If a signal was caught, send this signal to the application again and unblock only this signal
    so that it invokes the signal handler which was just reinstated in step 10.
13. Unblock all of the signals that were blocked in step 7.
Since the application is expected to handle signals in non-blocking server mode, `gl_get_line()` does not attempt to duplicate this when it is being called. If one of the signals that it is configured to catch is sent to the application while `gl_get_line()` is being called, `gl_get_line()` reinstates the caller's signal handlers, then immediately before returning, re-sends the signal to the process to let the application’s signal handler handle it. If the process is not terminated by this signal, `gl_get_line()` returns NULL, and a following call to `gl_return_status()` returns the enumerated value GLR_SIGNAL.

Often, rather than letting it terminate the process, applications respond to the SIGINT user-interrupt signal by aborting the current input line. This can be accomplished in non-blocking server-I/O mode by not calling `gl_handle_signal()` when this signal is caught, but by calling instead the `gl_abandon_line()` function. This function arranges that when `gl_get_line()` is next called, it first flushes any pending output to the terminal, discards the current input line, outputs a new prompt on the next line, and finally starts accepting input of a new input line from the user.

Provided that certain rules are followed, the `gl_normal_io()`, `gl_raw_io()`, `gl_handle_signal()`, and `gl_abandon_line()` functions can be written to be safely callable from signal handlers. Other functions in this library should not be called from signal handlers. For this to be true, all signal handlers that call these functions must be registered in such a way that only one instance of any one of them can be running at one time. The way to do this is to use the POSIX `sigaction()` function to register all signal handlers, and when doing this, use the `sa_mask` member of the corresponding `sigaction` structure to indicate that all of the signals whose handlers invoke the above functions should be blocked when the current signal is being handled. This prevents two signal handlers from operating on a GetLine object at the same time.

To prevent signal handlers from accessing a GetLine object while `gl_get_line()` or any of its associated public functions are operating on it, all public functions associated with `gl_get_line()`, including `gl_get_line()` itself, temporarily block the delivery of signals when they are accessing GetLine objects. Beware that the only signals that they block are the signals that `gl_get_line()` is currently configured to catch, so be sure that if you call any of the above functions from signal handlers, that the signals that these handlers are assigned to are configured to be caught by `gl_get_line()`. See `gl_trap_signal(3TECLA)`.

If instead of using `select()` or `poll()` to wait for I/O your application needs only to get out of `gl_get_line()` periodically to briefly do something else before returning to accept input from the user, use the `gl_inactivity_timeout(3TECLA)` function in non-blocking server mode to specify that a callback function that returns GLT0_CONTINUE should be called whenever `gl_get_line()` has been waiting for I/O for more than a specified amount of time. When this callback is triggered, `gl_get_line()` will return NULL and a following call to `gl_return_status()` will return GLR_BLOCKED.

The `gl_get_line()` function will not return until the user has not typed a key for the specified interval, so if the interval is long and the user keeps typing, `gl_get_line()` might not return
for a while. There is no guarantee that it will return in the time specified.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  cpl_complete_word(3TECLA), ef_expand_file(3TECLA), gl_get_line(3TECLA),
          libtecla(3LIB), pca_lookup_file(3TECLA), attributes(5), tecla(5)
hextob(3TSOL)

Name  hextob, htobsl, htobclear – convert hexadecimal string to binary label

Synopsis  cc [flag...] file... -ltsol [library...]

#include <tsol/label.h>

int htobsl(const char *s, m_label_t *label);
int htobclear(const char *s, m_label_t *clearance);

Description  These functions convert hexadecimal string representations of internal label values into binary labels.

The htobsl() function converts into a binary sensitivity label, a hexadecimal string of the form:

0x\text{sensitivity\_label\_hexadecimal\_value}

The htobclear() function converts into a binary clearance, a hexadecimal string of the form:

0x\text{clearance\_hexadecimal\_value}

Return Values  These functions return non-zero if the conversion was successful, otherwise zero is returned.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

These functions are obsolete and retained for ease of porting. They might be removed in a future Solaris Trusted Extensions release. Use the str_to_label(3TSOL) function instead.

See Also  libtsol(3LIB), str_to_label(3TSOL), attributes(5), labels(5)

Notes  The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
**Name**
lableclipping, Xbsltos, Xbcleartos – translate a binary label and clip to the specified width

**Synopsis**

c c [flag... ] file... -ltsol -ldTsol [library...]

```
#include <Dt/label_clipping.h>

XmString Xbsltos(Display *display, const m_label_t *senslabel,
               Dimension width, const XmFontList fontlist, const int flags);

XmString Xbcleartos(Display *display, const m_label_t *clearance,
               Dimension width, const XmFontList fontlist, const int flags);
```

**Description**
The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to translate labels or clearances that dominate the current process’ sensitivity label.

- **display**: The structure controlling the connection to an X Window System display.
- **senslabel**: The sensitivity label to be translated.
- **clearance**: The clearance to be translated.
- **width**: The width of the translated label or clearance in pixels. If the specified width is shorter than the full label, the label is clipped and the presence of clipped letters is indicated by an arrow. In this example, letters have been clipped to the right of: TS<-. See the sbltos(3TSOL) manual page for more information on the clipped indicator. If the specified width is equal to the display width (display), the label is not truncated, but word-wrapped using a width of half the display width.
- **fontlist**: A list of fonts and character sets where each font is associated with a character set.
- **flags**: The value of flags indicates which words in the label_encodings(4) file are used for the translation. See the bltos(3TSOL) manual page for a description of the flag values: LONG_WORDS, SHORT_WORDS, LONG_CLASSIFICATION, SHORT_CLASSIFICATION, ALL_ENTRIES, ACCESS_RELATED, VIEW_EXTERNAL, VIEW_INTERNAL, NO_CLASSIFICATION. BRACKETED is an additional flag that can be used with Xbsltos() only. It encloses the sensitivity label in square brackets as follows: [C].

**Return Values**
These functions return a compound string that represents the character-coded form of the sensitivity label or clearance that is translated. The compound string uses the language and fonts specified in fontlist and is clipped to width. These functions return NULL if the label or clearance is not a valid, required type as defined in the label_encodings(4) file, or not dominated by the process’ sensitivity label and the PRIV_SYS_TRANS_LABEL privilege is not asserted.

**Files**

```
/usr/dt/include/Dt/label_clipping.h
```

Header file for label clipping functions
/etc/security/tsol/label_encodings

The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**Examples**

**EXAMPLE 1** Translate and Clip a Clearance.

This example translates a clearance to text using the long words specified in the `label_encodings(4)` file, a font list, and clips the translated clearance to a width of 72 pixels.

```c
xmstr = Xbcleartos(XtDisplay(topLevel),
&clearance, 72, fontlist, LONG_WORDS)
```

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

The labelclipping functions, `Xbsltos()` and `Xbcleartos()`, are obsolete. Use the `label_to_str(3TSOL)` function instead.

**See Also**

`bltos(3TSOL), label_to_str(3TSOL), libtsol(3LIB), label_encodings(4), attributes(5)`

See `XmStringDraw(3)` and `FontList(3)` for information on the creation and structure of a font list.

**Notes**

The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
**Synopsis**

```
cc [flag...] file... -ltsol [library...]
#include <tsol/label.h>

int label_to_str(const m_label_t *label, char **string,
                 const m_label_str_t conversion_type, uint_t flags);
```

**Description**

`label_to_str()` is a simple function to convert various mandatory label types to human readable strings.

`label` is the mandatory label to convert. `string` points to memory that is allocated by `label_to_str()` that contains the converted string. The caller is responsible for calling `free(3C)` to free allocated memory.

The calling process must have mandatory read access to the resulting human readable string. Or the calling process must have the `sys_trans_label` privilege.

The `conversion_type` parameter controls the type of label conversion. Not all types of conversion are valid for all types of label:

- **M_LABEL**
  - Converts `label` to a human readable string based on its type.

- **M_INTERNAL**
  - Converts `label` to an internal text representation that is safe for storing in a public object. Internal conversions can later be parsed to their same value.

- **M_COLOR**
  - Converts `label` to a string that represents the color name that the administrator has associated with the label.

- **PRINTER_TOP_BOTTOM**
  - Converts `label` to a human readable string that is appropriate for use as the top and bottom label of banner and trailer pages in the Defense Intelligence Agency (DIA) encodings printed output schema.

- **PRINTER_LABEL**
  - Converts `label` to a human readable string that is appropriate for use as the banner page downgrade warning in the DIA encodings printed output schema.

- **PRINTER_CAVEATS**
  - Converts `label` to a human readable string that is appropriate for use as the banner page caveats section in the DIA encodings printed output schema.

- **PRINTER_CHANNELS**
  - Converts `label` to a human readable string that is appropriate for use as the banner page handling channels in the DIA encodings printed output schema.

The `flags` parameter provides a hint to the label conversion:

- **DEF_NAMES**
  - The default names are preferred.
**label_to_str(3TSOL)**

**SHORT_NAMES**  Short names are preferred where defined.

**LONG_NAMES**  Long names are preferred.

**Return Values**  Upon successful completion, the `label_to_str()` function returns 0. Otherwise, -1 is returned, `errno` is set to indicate the error and the string pointer is set to `NULL`.

**Errors**  The `label_to_str()` function will fail if:

- **EINVAL**  Invalid parameter.
- **ENOTSUP**  The system does not support label translations.
- **ENOMEM**  The physical limits of the system are exceeded by size bytes of memory which cannot be allocated.

**Attributes**  See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>See below.</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Standard</td>
<td>See below.</td>
</tr>
</tbody>
</table>

The `label_to_str()` function is Committed. The returned string is Not-an-Interface and is dependent on the specific `label_encodings` file. The conversion type `INTERNAL` is Uncommitted, but is always accepted as input to `str_to_label(3TSOL)`.

Conversion types that are relative to the DIA encodings schema are Standard. Standard is specified in `label_encodings(4)`.

**See Also**  `free(3C), libtsol(3LIB), str_to_label(3TSOL), label_encodings(4), attributes(5), labels(5)`

**Warnings**  A number of these conversions rely on the DIA label encodings schema. They might not be valid for other label schemata.

**Notes**  The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
libtecla_version() function queries for the version number of the library.

On return, this function records the three components of the libtecla version number in
*major, *minor, *micro. The formal meaning of the three components is as follows:

- **major**: Incrementing this number implies that a change has been made to the library’s
  public interface that makes it binary incompatible with programs that were linked
  with previous shared versions of libtecla.

- **minor**: This number is incremented by one whenever additional functionality, such as new
  functions or modules, are added to the library.

- **micro**: This number is incremented whenever modifications to the library are made that
  make no changes to the public interface, but which fix bugs and/or improve the
  behind-the-scenes implementation.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
libtecla(3LIB), attributes(5)
media_findname(3VOLMGT)

Name  media_findname – convert a supplied name into an absolute path name that can be used to access removable media

Synopsis  cc [ flag ... ] file ... -lvolmgt [ library ... ]
#include <volmgt.h>

char *media_findname(char *start);

Description  This function is obsolete. The management of removable media by the Volume Management feature, including vold, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL website. See hal(5). The return value of this function is undefined.

media_findname() converts the supplied start string into an absolute path name that can then be used to access a particular piece of media.

The start parameter can be one of the following types of specifications:

/dev/... An absolute path name in /dev, in which case a copy of that string is returned (see NOTES on this page).

volume_name The volume name for a particular volume, such as fred.

volmgt_symname The symbolic name for a device, such as cdrom2.

media_type The generic media type name, such as cdrom. In this case media_findname() looks for the first piece of media that matches that media type, starting at 0 (zero) and continuing on until a match is found (or some fairly large maximum number is reached). In this case, if a match is found, a copy of the path name to the volume found is returned.

Return Values  The return from this function is undefined.

Errors  For cases where the supplied start parameter is an absolute path name, media_findname() can fail, returning a null string pointer, if an lstat(2) of that supplied pathname fails. Also, if the supplied absolute pathname is a symbolic link, media_findname() can fail if a readlink(2) of that symbolic link fails, or if a stat(2) of the pathname pointed to by that symbolic link fails, or if any of the following is true:

ENXIO The specified absolute pathname was not a character special device, and it was not a directory with a character special device in it.

Examples  EXAMPLE 1  Sample programs of the media_findname() function.

The following example attempts to find what the pathname is to a piece of media called fred. Notice that a volmgt_check() is done first (see the NOTES section on this page).

(void) volmgt_check(NULL);
if ((nm = media_findname("fred")) != NULL) {


Sample programs of the `media_findname()` function.

```c
(void) printf("media named \"fred\" is at \"%s\"\n", nm);
} else {
    (void) printf("media named \"fred\" not found\n");
}
```

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Unsafe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

**See Also**  
`lstat(2), readlink(2), stat(2), free(3C), malloc(3C), volmgt_check(3VOLMGT), volmgt_inuse(3VOLMGT), volmgt_root(3VOLMGT), volmgt_running(3VOLMGT), volmgt_symname(3VOLMGT), attributes(5), hal(5)`

**Notes**  
If `media_findname()` cannot find a match for the supplied name, it performs a `volmgt_check(3VOLMGT)` and tries again, so it can be more efficient to perform `volmgt_check()` before calling `media_findname()`.

Upon success `media_findname()` returns a pointer to string which has been allocated; this should be freed when no longer in use (see `free(3C)`).
media_getattr, media_setattr – get and set media attributes

Synopsis

```
cc [ flag ... ] file ... -lvolmgt [ library ... ]
```

```
#include <volmgt.h>

char *media_getattr(char *vol_path, char *attr);
int media_setattr(char *vol_path, char *attr, char *value);
```

Description

This function is obsolete. The management of removable media by the Volume Management feature, including vold, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL website. See `hal(5)`. The return value of this function is undefined.

`media_setattr()` and `media_getattr()` respectively set and get attribute-value pairs (called properties) on a per-volume basis.

Volume management supports system properties and user properties. System properties are ones that volume management predefines. Some of these system properties are writable, but only by the user that owns the volume being specified, and some system properties are read only:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Writable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-access</td>
<td>RO</td>
<td>“seq”, “rand”</td>
<td>sequential or random access</td>
</tr>
<tr>
<td>s-density</td>
<td>RO</td>
<td>“low”, “medium”, “high”</td>
<td>media density</td>
</tr>
<tr>
<td>s-parts</td>
<td>RO</td>
<td>comma separated list of slice numbers</td>
<td>list of partitions on this volume</td>
</tr>
<tr>
<td>s-location</td>
<td>RO</td>
<td><code>pathname</code></td>
<td>volume management path name to media</td>
</tr>
<tr>
<td>s-mejectable</td>
<td>RO</td>
<td>“true”, “false”</td>
<td>whether or not media is manually ejectable</td>
</tr>
<tr>
<td>s-rmoneject</td>
<td>R/W</td>
<td>“true”, “false”</td>
<td>should media access points be removed from database upon ejection</td>
</tr>
<tr>
<td>s-enxio</td>
<td>R/W</td>
<td>“true”, “false”</td>
<td>if set return ENXIO when media access attempted</td>
</tr>
</tbody>
</table>

Properties can also be defined by the user. In this case the value can be any string the user wishes.

Return Values

The return from this function is undefined.

Errors

Both `media_getattr()` and `media_setattr()` can fail returning a null pointer if an `open(2)` of the specified `vol_path` fails, if an `fstat(2)` of that pathname fails, or if that pathname is not a block or character special device.
media_getattr() can also fail if the specified attribute was not found, and media_setattr() can also fail if the caller doesn't have permission to set the attribute, either because it's a system attribute, or because the caller doesn't own the specified volume.

**Examples**

**EXAMPLE1** Using `media_getattr()`

The following example checks to see if the volume called `fred` that volume management is managing can be ejected by means of software, or if it can only be manually ejected:

```c
if (media_getattr("/rdsk/fred", "s-mejectable") != NULL) {
    (void) printf("\"fred\" must be manually ejected\n");
} else {
    (void) printf("software can eject \"fred\"\n");
}
```

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

**See Also**

`lstat(2), open(2), readlink(2), stat(2), free(3C), malloc(3C), media_findname(3VOLMGT), volmgt_check(3VOLMGT), volmgt_inuse(3VOLMGT), volmgt_root(3VOLMGT), volmgt_running(3VOLMGT), volmgt_symname(3VOLMGT), attributes(5), hal(5)`
media_getid(3VOLMGT)

Name  media_getid – return the id of a piece of media

Synopsis  cc [flag ...] file ...-lvolgmt [library ...]

#include <volmgmt.h>

ulonglong_t media_getid(char *vol_path);

Description  This function is obsolete. The management of removable media by the Volume Management feature, including vold, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL website. See hal(5). The return value of this function is undefined.

media_getid() returns the id of a piece of media. Volume management must be running. See volmgmt_running(3VOLMGT).

Parameters  vol_path  Path to the block or character special device.

Return Values  The return from this function is undefined.

Examples  EXAMPLE 1  Using media_getid()

The following example first checks if volume management is running, then checks the volume management name space for path, and then returns the id for the piece of media.

char *path;
...

if (volmgmt_running()) {
    if (volmgmt_owns_path(path)) {
        (void) printf("id of %s is %lld\n", path, media_getid(path));
    }
}

If a program using media_getid() does not check whether or not volume management is running, then any NULL return value will be ambiguous, as it could mean that either volume management does not have path in its name space, or volume management is not running.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>
See Also volmgt_owndpath(3VOLMGT), volmgt_running(3VOLMGT), attributes(5), hal(5)
m_label(3TSOL)

**Name**  
m_label, m_label_alloc, m_label_dup, m_label_free – m_label functions

**Synopsis**  
cc [flag...] file... -ltsol [library...]  
#include <tsol/label.h>  
m_label_t *m_label_alloc(const m_label_type_t label_type);  
int m_label_dup(m_label_t **dst, const m_label_t *src);  
void m_label_free(m_label_t *label);

**Description**  
The `m_label_alloc()` function allocates resources for a new label. The `label_type` argument defines the type for a newly allocated label. The label type can be:

- MAC_LABEL - A Mandatory Access Control (MAC) label.
- USER_CLEAR - A user clearance.

The `m_label_dup()` function allocates resources for a new `dst` label. The function returns a pointer to the allocated label, which is an exact copy of the `src` label. The caller is responsible for freeing the allocated resources by calling `m_label_free()`.

The `m_label_free()` function frees resources that are associated with the previously allocated label.

**Return Values**  
Upon successful completion, the `m_label_alloc()` function returns a pointer to the newly allocated label. Otherwise, `m_label_alloc()` returns NULL and `errno` is set to indicate the error.

Upon successful completion, the `m_label_dup()` function returns 0. Otherwise, -1 is returned and `errno` is set to indicate the error.

**Errors**  
The `m_label_alloc()` function will fail if:

- EINVAL - Invalid parameter.
- ENOMEM - The physical limits of the system are exceeded by `size` bytes of memory which cannot be allocated.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
label_to_str(3TSOL), libtsol(3LIB), str_to_label(3TSOL), label_encodings(4), attributes(5), labels(5)
**Notes**  The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
The PathCache object is part of the libtecla(3LIB) library. PathCache objects allow an application to search for files in any colon separated list of directories, such as the UNIX execution PATH environment variable. Files in absolute directories are cached in a PathCache object, whereas relative directories are scanned as needed. Using a PathCache object, you can look up the full pathname of a simple filename, or you can obtain a list of the possible completions of a given filename prefix. By default all files in the list of directories are targets for lookup and completion, but a versatile mechanism is provided for only selecting specific types of files. The obvious application of this facility is to provide Tab-completion and lookup of executable commands in the UNIX PATH, so an optional callback which rejects all but executable files, is provided.

An Example

Under UNIX, the following example program looks up and displays the full pathnames of each of the command names on the command line.

```c
#include <stdio.h>
#include <stdlib.h>
#include <libtecla.h>

int main(int argc, char *argv[])
{
    int i;
    /*
```
* Create a cache for executable files.
 */
 PathCache *pc = new_PathCache();
 if(!pc)
   exit(1);
 /*
 * Scan the user’s PATH for executables.
 */
 if(pca_scan_path(pc, getenv("PATH"))) {
   fprintf(stderr, "%s\n", pca_last_error(pc));
   exit(1);
 }
 /*
 * Arrange to only report executable files.
 */
 pca_set_check_fn(pc, cpl_check_exe, NULL);
 /*
 * Lookup and display the full pathname of each of the
 * commands listed on the command line.
 */
 for(i=1; i<argc; i++) {
   char *cmd = pca_lookup_file(pc, argv[i], -1, 0);
   printf("The full pathname of '%s' is %s\n", argv[i],
          cmd ? cmd : "unknown");
 }
 pc = del_PathCache(pc); /* Clean up */
 return 0;
}

The following is an example of what this does on a laptop under LINUX:

```
$ ./example less more blob
The full pathname of 'less' is /usr/bin/less
The full pathname of 'more' is /bin/more
The full pathname of 'blob' is unknown
```

Function Descriptions

To use the facilities of this module, you must first allocate a PathCache object by calling the new_PathCache() constructor function. This function creates the resources needed to cache and lookup files in a list of directories. It returns NULL on error.

Populating The Cache

Once you have created a cache, it needs to be populated with files. To do this, call the pca_scan_path() function. Whenever this function is called, it discards the current contents of the cache, then scans the list of directories specified in its path argument for files. The path argument must be a string containing a colon-separated list of directories, such as “/usr/bin:/home/mcs/bin:”. This can include directories specified by absolute pathnames such as “/usr/bin”, as well as sub-directories specified by relative pathnames such as “.” or “bin”. Files in the absolute directories are immediately cached in the specified PathCache.
object, whereas subdirectories, whose identities obviously change whenever the current
working directory is changed, are marked to be scanned on the fly whenever a file is looked up.

On success this function return 0. On error it returns 1, and a description of the error can be
obtained by calling pca_last_error(pc).

Looking Up Files

Once the cache has been populated with files, you can look up the full pathname of a file,
simply by specifying its filename to pca_lookup_file().

To make it possible to pass this function a filename which is actually part of a longer string, the
dependent argument can be used to specify the length of the filename at the start of the name[]
argument. If you pass -1 for this length, the length of the string will be determined with strlen.
If the name[] string might contain backslashes that escape the special meanings of spaces and
tabs within the filename, give the literal argument the value 0. Otherwise, if backslashes should
be treated as normal characters, pass 1 for the value of the literal argument.

Filename Completion

Looking up the potential completions of a filename-prefix in the filename cache is achieved by
passing the provided pca_path_completions() callback function to the
cpl_complete_word(3TECLA) function.

This callback requires that its data argument be a pointer to a PcaPathConf object.
Configuration objects of this type are allocated by calling new_PcaPathConf().

This function returns an object initialized with default configuration parameters, which
determine how the cpl_path_completions() callback function behaves. The functions which
allow you to individually change these parameters are discussed below.

By default, the pca_path_completions() callback function searches backwards for the start of
the filename being completed, looking for the first un-escaped space or the start of the input
line. If you wish to specify a different location, call ppc_file_start() with the index at which
the filename starts in the input line. Passing start_index=-1 re-enables the default behavior.

By default, when pca_path_completions() looks at a filename in the input line, each lone
backslash in the input line is interpreted as being a special character which removes any
special significance of the character which follows it, such as a space which should be taken as
part of the filename rather than delimiting the start of the filename. These backslashes are thus
ignored while looking for completions, and subsequently added before spaces, tabs and literal
backslashes in the list of completions. To have unescaped backslashes treated as normal
characters, call ppc_literal_escapes() with a non-zero value in its literal argument.

When you have finished with a PcaPathConf variable, you can pass it to the
del_PcaPathConf() destructor function to reclaim its memory.

Being Selective

If you are only interested in certain types or files, such as, for example, executable files, or files
whose names end in a particular suffix, you can arrange for the file completion and lookup
functions to be selective in the filenames that they return. This is done by registering a callback
function with your PathCache object. Thereafter, whenever a filename is found which either
matches a filename being looked up or matches a prefix which is being completed, your
callback function will be called with the full pathname of the file, plus any application-specific
data that you provide. If the callback returns 1 the filename will be reported as a match. If it
returns 0, it will be ignored. Suitable callback functions and their prototypes should be
declared with the following macro. The CplCheckFn typedef is also provided in case you wish
to declare pointers to such functions

#define CPL_CHECK_FN(fn) int (fn)(void *data, const char *pathname)
typedef CPL_CHECK_FN(CplCheckFn);

Registering one of these functions involves calling the pca_set_check_fn() function. In
addition to the callback function passed with the check_fn argument, you can pass a pointer to
anything with the data argument. This pointer will be passed on to your callback function by
its own data argument whenever it is called, providing a way to pass application-specific data
to your callback. Note that these callbacks are passed the full pathname of each matching file,
so the decision about whether a file is of interest can be based on any property of the file, not
just its filename. As an example, the provided cpl_check_exe() callback function looks at the
executable permissions of the file and the permissions of its parent directories, and only
returns 1 if the user has execute permission to the file. This callback function can thus be used
to lookup or complete command names found in the directories listed in the user’s PATH
environment variable. The example program above provides a demonstration of this.

Beware that if somebody tries to complete an empty string, your callback will get called once
for every file in the cache, which could number in the thousands. If your callback does
anything time consuming, this could result in an unacceptable delay for the user, so callbacks
should be kept short.

To improve performance, whenever one of these callbacks is called, the choice that it makes is
cached, and the next time the corresponding file is looked up, instead of calling the callback
again, the cached record of whether it was accepted or rejected is used. Thus if somebody tries
to complete an empty string, and hits tab a second time when nothing appears to happen,
there will only be one long delay, since the second pass will operate entirely from the cached
dispositions of the files. These cached dispositions are discarded whenever pca_scan_path() is
called, and whenever pca_set_check_fn() is called with changed callback function or data
arguments.

Error Handling If pca_scan_path() reports that an error occurred by returning 1, you can obtain a terse
description of the error by calling pca_last_error(pc). This returns an internal string
containing an error message.

Cleaning Up Once you have finished using a PathCache object, you can reclaim its resources by passing it to
the del_PathCache() destructor function. This takes a pointer to one of these objects, and
always returns NULL.
It is safe to use the facilities of this module in multiple threads, provided that each thread uses a separately allocated PathCache object. In other words, if two threads want to do path searching, they should each call new PathCache() to allocate their own caches.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  cpl_complete_word(3TECLA), ef_expand_file(3TECLA), gl_get_line(3TECLA), libtecla(3LIB), attributes(5)
sbltos, sbsltos, sbcleartos – translate binary labels to canonical character-coded labels

**Synopsis**
```
cc [flag...] file... -ltsol [library...]
#include <tsol/label.h>

char *sbltos(const m_label_t *label, const int len);
char *sbcleartos(const m_label_t *clearance, const int len);
```

**Description**
These functions translate binary labels into canonical strings that are clipped to the number of printable characters specified in `len`. Clipping is required if the number of characters of the translated string is greater than `len`. Clipping is done by truncating the label on the right to two characters less than the specified number of characters. A clipped indicator, "<−", is appended to sensitivity labels and clearances. The character-coded label begins with a classification name separated with a single space character from the list of words making up the remainder of the label. The binary labels must be of the proper defined type and dominated by the process's sensitivity label. A `len` of 0 (zero) returns the entire string with no clipping.

The `sbltos()` function translates a binary sensitivity label into a clipped string using the long form of the words and the short form of the classification name. If `len` is less than the minimum number of characters (three), the translation fails.

The `sbcleartos()` function translates a binary clearance into a clipped string using the long form of the words and the short form of the classification name. If `len` is less than the minimum number of characters (three), the translation fails. The translation of a clearance might not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file which might contain different words and constraints.

The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to perform label translation on labels that dominate the current process's sensitivity label.

**Process Attributes**
If the VIEW_EXTERNAL or VIEW_INTERNAL flags are not specified, translation of `ADMIN_LOW` and `ADMIN_HIGH` labels is controlled by the label view process attribute flags. If no label view process attribute flags are defined, their translation is controlled by the label view configured in the `label_encodings` file. A value of External specifies that `ADMIN_LOW` and `ADMIN_HIGH` labels are mapped to the lowest and highest labels defined in the `label_encodings` file. A value of Internal specifies that the `ADMIN_LOW` and `ADMIN_HIGH` labels are translated to the `admin low name` and `admin high name` strings specified in the `label_encodings` file. If no such names are specified, the strings "ADMIN_LOW" and "ADMIN_HIGH" are used.

**Return Values**
These functions return a pointer to a statically allocated string that contains the result of the translation, or (char *)0 if the translation fails for any reason.

**Examples**
sbltos() Assume that a sensitivity label is:
UN TOP/MIDDLE/LOWER DRAWER
When clipped to ten characters it is:
UN TOP/M<-

sbcleartos() Assume that a clearance is:
UN TOP/MIDDLE/LOWER DRAWER
When clipped to ten characters it is:
UN TOP/M<-

Files /etc/security/tsol/label_encodings
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

These functions are obsolete and retained for ease of porting. They might be removed in a future Solaris Trusted Extensions release. Use the label_to_str(3TSOL) function instead.

See Also label_to_str(3TSOL), libtsol(3LIB), attributes(5), labels(5)

Warnings All these functions share the same statically allocated string storage. They are not MT-Safe. Subsequent calls to any of these functions will overwrite that string with the newly translated string.

Notes The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
Name  

scf_decoration_create, scf_decoration_handle, scf_decoration_destroy,
scf_service_get_decoration, scf_instance_get_decoration, scf_pg_get_decoration,
scf_property_get_decoration, scf_property_get_value_at_layer, scf_iter_service_decorations,
scf_iter_instance_decorations, scf_iter_pg_decorations, scf_iter_property_decorations,
scf_iter_decoration_values, scf_iter_next_decoration, scf_decoration_get_bundle,
scf_decoration_get_layer, scf_decoration_layer_to_string,
scf_decoration_layer_from_string, scf_decoration_is_type, scf_decoration_type,
scf_decoration_get_value – get decoration and layer information in the Service Configuration Facility

Synopsis  

cc [ flag... ] file... -lsacf [ library... ]
#include <libscf.h>

scf_decoration_t *scf_decoration_create(scf_handle_t *handle);
scf_handle_t *scf_decoration_handle(const scf_decoration_t *dec);
void scf_decoration_destroy(scf_decoration_t *dec);
int scf_service_get_decoration(const scf_service_t *,
scf_decoration_layer_t, scf_decoration_t *dec);
int scf_instance_get_decoration(const scf_instance_t *
inst,
scf_decoration_layer_t layer, scf_decoration_t *dec);
int scf_pg_get_decoration(const scf_propertygroup_t *pg,
scf_decoration_layer_t layer, scf_decoration_t *dec);
int scf_property_get_decoration(const scf_property_t *
prop,
scf_decoration_layer_t layer, scf_decoration_t *dec);
int scf_property_get_value_at_layer(const scf_property_t *
prop,
scf_decoration_layer_t layer, scf_value_t *v);
int scf_iter_service_decorations(scf_iter_t *iter,
const scf_service_t *svc, int flags);
int scf_iter_instance_decorations(scf_iter_t *iter,
const scf_instance_t *inst, int flags);
int scf_iter_pg_decorations(scf_iter_t *iter,
const scf_propertygroup_t *pg, int flags);
int scf_iter_property_decorations(scf_iter_t *iter,
const scf_property_t *);
int scf_iter_decoration_values(scf_iter_t *,
const scf_decoration_t *dec);
int scf_iter_next_decoration(scf_iter_t *iter, scf_decoration_t *out);
ssize_t scf_decoration_get_bundle(const scf_decoration_t *dec,
char *out, size_t len);
int scf_decoration_get_layer(const scf_decoration_t *dec,
    scf_decoration_layer_t *out);

const char *scf_decoration_layer_to_string(scf_decoration_layer_t layer);
scf_decoration_layer_t scf_decoration_layer_from_string(const char *layer);
int scf_decoration_is_type(const scf_decoration_t *dec, scf_type_t base_arg);
int scf_decoration_type(const scf_decoration_t *dec, scf_type_t *out);
int scf_decoration_get_value(const scf_decoration_t *dec,
    scf_value_t *v);

Description
The scf_decoration_*() functions allow libscf(3LIB) consumers to explore
decorations of each type of libscf entity. Basic_get interfaces are provided, along with
iterators for all decorations on an entity.

Decorations are metadata about entities within libscf. A single entity may have multiple
decorations in the repository to describe metadata about what files or actions contributed to
that entity. These decorations are disambiguated by their layer. Only four layers are used,
though more may be added later: admin, site-profile, system-profile, and manifest.

These layers are ordered, with the admin layer overriding the site-profile layer, and down
to the manifest layer as the lowest layer.

All property reading libscf interfaces will get the value of properties contributed only by the
highest layer. All property writing libscf interfaces will change values in only the
administrative layer. The interfaces documented here allow access to layers explicitly.

Additionally, decorations are completely general, and in the future non-layer decorations may
be added to capture other metadata.

An scf_decoration_t is an opaque handle that can be set to a single decoration at any given
time. The scf_decoration_create() function allocates and initializes a new
scf_decoration_t bound to a handle. The scf_decoration_destroy() function destroys
and frees the decoration. The scf_decoration_handle() function retrieves the handle to
which the decoration is bound.

The scf_decoration_layer_t describes specific layers:

- SCF_DECORATION_ADMIN is the admin layer.
- SCF_DECORATION_SITE_PROFILE is the site-profile layer.
- SCF_DECORATION_SYSTEM_PROFILE is the system-profile layer.
- SCF_DECORATION_MANIFEST is the manifest layer.

The scf_service_get_decoration(), scf_instance_get_decoration(),
scf_pg_get_decoration(), and scf_property_get_decoration() functions take an entity
and an scf_decoration_layer_t in order to retrieve the decoration at the requested layer.
The layer may also be defined as SCF_DECORATION_TOP to get the topmost layer.
The `scf_property_get_value_at_layer()` function retrieves the `scf_value_t` at the defined layer.

The `scf_iter_service_decorations()`, `scf_iter_instance_decorations()`, `scf_iter_pg_decorations()`, `scf_iter_property_decorations()`, and `scf_iter_decoration_values()` functions take an iterator created by `scf_iter_create(3SCF)` and start the iteration over all decorations on the supplied entity. The `scf_iter_next_decoration()` function gets the next decoration.

A set of functions offer access to information about a retrieved decoration. The `scf_decoration_get_bundle()` function retrieves the file, if any, that a decoration was created from.

The `scf_decoration_get_layer()` and `scf_decoration_layer_from_string()` functions retrieve the layer information. The `scf_decoration_layer_to_string()` function retrieves the layer information in string form.

Values within a decoration, if any, and their types can be retrieved with `scf_decoration_is_type()`, `scf_decoration_type()`, and `scf_decoration_get_value()`.

Upon successful completion, `scf_decoration_create()` returns a new `scf_property_t`. Otherwise, it returns `NULL`.

Upon successful completion, `scf_decoration_handle()` returns an `scf_handle_t`. Otherwise, it returns `-1`.

Upon successful completion, `scf_service_get_decoration()`, `scf_instance_get_decoration()`, `scf_pg_get_decoration()`, `scf_property_get_decoration()`, and `scf_property_get_value_at_layer()` return 0. Otherwise, they return `-1`.

Upon successful completion, `scf_iter_service_decorations()`, `scf_iter_instance_decorations()`, `scf_iter_pg_decorations()`, `scf_iter_property_decorations()`, and `scf_iter_decoration_values()` return 0. Otherwise, they return `-1`.

Upon successful completion, `scf_iter_next_decoration()` returns 1. If the iterator is complete, it returns 0. Otherwise, it returns `-1`.

Upon successful completion `scf_decoration_get_bundle()` returns the length of the string written, not including the terminating null byte. Otherwise, it returns `-1`.

Upon successful completion, `scf_decoration_get_layer()`, `scf_decoration_type()`, `scf_decoration_get_value()`, and `scf_decoration_is_type()` return 0. On error, they return `-1`.

Errors The `scf_decoration_create()` function will fail if:

- `SCF_ERROR_INVALID_ARGUMENT` The value of the `handle` argument is `NULL`.
SCF_ERROR_NO_MEMORY
    There is not enough memory to allocate an scf_decoration_t.

SCF_ERROR_NO_RESOURCES
    The server does not have adequate resources for a new property handle.

The scf_decoration_handle() function will fail if:

SCF_ERROR_HANDLE_DESTROYED
    The handle associated with prop has been destroyed.

The scf_service_get_decoration(), scf_instance_get_decoration(),
scf_pg_get_decoration(), and scf_property_get_decoration() functions will fail if:

SCF_ERROR_CONSTRAINT_VIOLATED
    Multiple decorations are available at the specified layer.

The scf_service_get_decoration(), scf_instance_get_decoration(),
scf_pg_get_decoration(), scf_property_get_decoration(),
scf_property_get_value_at_layer(), scf_decoration_get_bundle(), and
scf_decoration_get_value() functions will fail if:

SCF_ERROR_BACKEND_ACCESS
    The storage mechanism that the repository server (svc.configd(1M)) chose for the
    operation denied access.

SCF_ERROR_CONNECTION_BROKEN
    The connection to the repository was lost.

SCF_ERROR_DELETED
    The entity or an ancestor has been deleted.

SCF_ERROR_HANDLE_MISMATCH
    The decoration and entity are not derived from the same handle.

SCF_ERROR_INTERNAL
    An internal error occurred.

SCF_ERROR_INVALID_ARGUMENT
    The entity is not the correct type.

SCF_ERROR_NO_RESOURCES
    The server does not have the resources to complete the request.

SCF_ERROR_NOT_BOUND
    The handle was never bound or has been unbound.

SCF_ERROR_NOT_SET
    The entity is not set.
The `scf_iter_service_decorations()`, `scf_iter_instance_decorations()`, `scf_iter_pg_decorations()`, `scf_iter_property_decorations()`, `scf_iter_decoration_values()`, and `scf_iter_next_decoration()` functions will fail if:

- **SCF_ERROR_DELETED**
  The parent has been deleted.
- **SCF_ERROR_NOT_SET**
  The parent is not set.
- **SCF_ERROR_NOT_BOUND**
  The handle is not bound.
- **SCF_ERROR_NO_RESOURCES**
  The server does not have the resources to complete the request.
- **SCF_ERROR_CONNECTION_BROKEN**
  The connection to the repository was lost.
- **SCF_ERROR_HANDLE_MISMATCH**
  The `iter` and `parent` arguments are not derived from the same handle.

The `scf_iter_next_decoration()` function will fail if:

- **SCF_ERROR_INVALID_ARGUMENT**
  The requested object type does not match the type the iterator is walking.

The `scf_decoration_get_layer()`, `scf_decoration_type()`, and `scf_decoration_is_type()` functions will fail if:

- **SCF_ERROR_NOT_BOUND**
  The handle was never bound or has been unbound.
- **SCF_ERROR_DELETED**
  The parent has been deleted.
- **SCF_ERROR_NOT_SET**
  The parent is not set.
- **SCF_ERROR_INTERNAL**
  An internal error occurred.
- **SCF_ERROR_CONNECTION_BROKEN**
  The connection to the repository was lost.

The `scf_decoration_is_type()` function will fail if:

- **SCF_ERROR_INVALID_ARGUMENT**
  The `type` argument is not a valid type.
- **SCF_ERROR_TYPE_MISMATCH**
  The `prop` argument is not of a type compatible with `type`. 
The \texttt{scf\_error(3SCF)} function can be used to retrieve the error value.

Attributes  See \texttt{attributes(5)} for descriptions of the following attributes:

\begin{tabular}{|l|l|}
  \hline
  \textbf{ATTRIBUTE TYPE} & \textbf{ATTRIBUTE VALUE} \\
  \hline
  Interface Stability & Committed \\
  MT-Level & Safe \\
  \hline
\end{tabular}

See Also  \texttt{svc\_configd(1M), libscf(3LIB), scf\_error(3SCF), scf\_iter\_create(3SCF), attributes(5)}
The `scf_entry_create()` function allocates a new transaction entry handle. The `scf_entry_destroy()` function destroys the transaction entry handle.

The `scf_entry_handle()` function retrieves the handle associated with `entry`.

A transaction entry represents a single action on a property in a property group. If an entry is added to a transaction using `scf_transaction_property_new(3SCF)`, `scf_transaction_property_change(3SCF)`, or `scf_transaction_property_change_type(3SCF)`, `scf_entry_add_value()` can be called zero or more times to set up the set of values for that property. Each value must be set and of a compatible type to the type associated with the entry. When later retrieved from the property, the values will have the type of the entry. If the values are committed successfully with `scf_transaction_commit(3SCF)`, they will be set in the order in which they were added with `scf_entry_add_value()`.

The `scf_entry_reset()` function resets a transaction entry, disassociating it from any transaction it is a part of (invalidating the transaction in the process), and disassociating any values that were added to it.

The `scf_entry_destroy_children()` function destroys all values associated with the transaction entry. The entry itself is not destroyed.

Upon successful completion, `scf_entry_create()` returns a new `scf_transaction_entry_t`. Otherwise, it returns NULL.

Upon successful completion, `scf_entry_handle()` returns the handle associated with the transaction entry. Otherwise, it returns NULL.

Upon successful completion, `scf_entry_add_value()` returns 0. Otherwise, it returns -1.
The `scf_entry_create()` function will fail if:

- **SCF_ERROR_INVALID_ARGUMENT**
  - The `handle` argument is NULL.

- **SCF_ERROR_NO_MEMORY**
  - There is not enough memory to allocate an `scf_transaction_entry_t`.

The `scf_entry_handle()` function will fail if:

- **SCF_ERROR_HANDLE_DESTROYED**
  - The handle associated with entry has been destroyed.

The `scf_entry_add_value()` function will fail if:

- **SCF_ERROR_HANDLE_MISMATCH**
  - The `value` and `entry` arguments are not derived from the same handle.

- **SCF_ERROR_IN_USE**
  - The value has been added to another entry.

- **SCF_ERROR_INTERNAL**
  - An internal error occurred.

- **SCF_ERROR_INVALID_ARGUMENT**
  - The `value` argument is not set, or the entry was added to the transaction using `scf_transaction_property_delete(3SCF)`.

- **SCF_ERROR_NOT_SET**
  - The transaction entry is not associated with a transaction.

- **SCF_ERROR_TYPE_MISMATCH**
  - The type of the `value` argument does not match the type that was set using `scf_transaction_property_new()`, `scf_transaction_property_change()`, or `scf_transaction_property_change_type()`.

The `scf_error(3SCF)` function can be used to retrieve the error value.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**

`libscf(3LIB)`, `scf_error(3SCF)`, `scf_transaction_commit(3SCF)`,
`scf_transaction_property_change(3SCF)`,
`scf_transaction_property_change_type(3SCF)`,
`scf_transaction_property_delete(3SCF)`, `scf_transaction_property_new(3SCF)`,
Name: scf_error, scf_strerror – error interface to Service Configuration Facility

Synopsis:

    cc [ flag... ] file... \ libscf [ library... ]
    #include <libscf.h>

    scf_error_t scf_error(void);
    const char *scf_strerror(scf_error_t error);

Description:

The scf_error() function returns the current libscf(3LIB) error value for the current thread. If the immediately previous call to a libscf function failed, the error value will reflect the reason for that failure.

The scf_strerror() function takes an error code previously returned by scf_error() and returns a human-readable, localized description of the error.

The error values are as follows:

- **SCF_ERROR_BACKEND_ACCESS**
  The storage mechanism that the repository server (svc.configd(1M)) chose for the operation denied access.

- **SCF_ERROR_BACKEND_READONLY**
  The storage mechanism that the repository server (svc.configd) chose for the operation is read-only. For the local filesystem storage mechanism (currently /etc/svc/repository.db), this usually occurs because the filesystem that contains it is mounted read-only. See mount(1M)

- **SCF_ERROR_CONNECTION_BROKEN**
  The connection to repository is broken.

- **SCF_ERROR_CONSTRAINT_VIOLATED**
  A required constraint was not met.

- **SCF_ERROR_DELETED**
  Object was deleted.

- **SCF_ERROR.Exists**
  The object already exists.

- **SCF_ERROR_HANDLE_DESTROYED**
  An object was bound to a destroyed handle.

- **SCF_ERROR_HANDLE_MISMATCH**
  Objects from different SCF handles were used.

- **SCF_ERROR_IN_USE**
  The object is currently in use.

- **SCF_ERROR_INTERNAL**
  An internal error occurred.

- **SCF_ERROR_INVALID_ARGUMENT**
  An argument is invalid.

- **SCF_ERROR_NO_MEMORY**
  No memory is available.

- **SCF_ERROR_NO_RESOURCES**
  The repository server is out of resources.

- **SCF_ERROR_NO_SERVER**
  The repository server is unavailable.

- **SCF_ERROR_NONE**
  No error occurred.
The `scf_error()` function returns `SCF_ERROR_NONE` if there have been no calls from `libscf` functions from the current thread. The return value is undefined if the immediately previous call to a `libscf` function did not fail.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

`svc.configd(1M), libscf(3LIB), attributes(5), svc.configd(1M)`
**Name**

`scf_handle_create`, `scf_handle_destroy`, `scf_handle_decorate`, `scf_handle_bind`, `scf_handle_unbind`, `scf_myname` - Service Configuration Facility handle functions

**Synopsis**

```c
cc [ flag... ] file... -lscf [ library... ]
#include <libscf.h>

scf_handle_t *scf_handle_create(scf_version_t version);
void scf_handle_destroy(scf_handle_t *handle);
int scf_handle_decorate(scf_handle_t *handle, const char *param,
   scf_value_t *value);
int scf_handle_bind(scf_handle_t *handle);
int scf_handle_unbind(scf_handle_t *handle);
ssize_t scf_myname(scf_handle_t *handle, char *out, size_t sz);
```

**Description**

The `scf_handle_create()` function creates a new Service Configuration Facility handle that is used as the base for all communication with the configuration repository. The version argument must be `SCF_VERSION`.

The `scf_handle_decorate()` function sets a single connection-level parameter, `param`, to the supplied value. If `value` is `SCF_DECORATE_CLEAR`, `param` is reset to its default state. Values passed to `scf_handle_decorate()` can be reset, reused, or destroyed. The values set do not take effect until `scf_handle_bind()` is called. Any invalid values will not cause errors prior to the call to `scf_handle_bind()`. The available decorations are:

- **debug** (count) Set the debugging flags.
- **show-masked** (boolean) This connection to the repository should show entities masked by the administrator. See `scf_service_is_masked(3SCF)` for details.

The `scf_handle_bind()` function binds the handle to a running `svc.configd(1M)` daemon, using the current decorations to modify the connection. All states derived from the handle are reset immediately after a successful binding.

The `scf_handle_unbind()` function severs an existing repository connection or clears the in-client state for a broken connection.

The `scf_handle_destroy()` function destroys and frees an SCF handle. It is illegal to use the handle after calling `scf_handle_destroy()`. Actions on subordinate objects act as if the handle is unbound.

The `scf_myname()` function retrieves the FMRI for the service of which the connecting process is a part. If the full FMRI does not fit in the provided buffer, it is truncated and, if `sz` > 0, zero-terminated.
Upon successful completion, `scf_handle_create()` returns the new handle. Otherwise, it returns `NULL`.

Upon successful completion, `scf_handle_decorate()`, `scf_handle_bind()`, and `scf_handle_unbind()` return 0. Otherwise, they return -1.

The `scf_mynname()` function returns the length of the full FMRI. Otherwise, it returns -1.

**Return Values**

Upon successful completion, `scf_handle_create()` returns the new handle. Otherwise, it returns `NULL`.

**Errors**

The `scf_handle_create()` function will fail if:

- **SCF_ERROR_NO_MEMORY**
  - There is no memory available.

- **SCF_ERROR_VERSION_MISMATCH**
  - The version is invalid, or the application was compiled against a version of the library that is more recent than the one on the system.

The `scf_handle_decorate()` function will fail if:

- **SCF_ERROR_INVALID_ARGUMENT**
  - The `param` argument is not a recognized parameter.

- **SCF_ERROR_TYPE_MISMATCH**
  - The `value` argument does not match the expected type for `param`.

- **SCF_ERROR_NOT_SET**
  - The `value` argument is not set.

The `scf_handle_bind()` function will fail if:

- **SCF_ERROR_INVALID_ARGUMENT**
  - One of the decorations was invalid.

- **SCF_ERROR_NO_SERVER**
  - The repository server is not running.

- **SCF_ERROR_NO_RESOURCES**
  - The server does not have adequate resources for a new connection.

The `scf_handle_unbind()` function will fail if:

- **SCF_ERROR_NOT_BOUND**
  - The handle is not bound.
The `scf_handle_mynamer()` function will fail if:

- **SCF_ERROR_CONNECTION_BROKEN**
  
  The connection to the repository was lost.

- **SCF_ERROR_NOT_BOUND**
  
  The handle is not bound.

- **SCF_ERROR_NOT_SET**
  
  This process is not marked as a SMF service.

The `scf_error(3SCF)` function can be used to retrieve the error value.

**Attributes**  
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>See below</td>
</tr>
</tbody>
</table>

Operations on a single handle (and the objects associated with it) are Safe. Operations on different handles are MT-Safe. Objects associated with different handles cannot be mixed, as this will lead to an `SCF_ERROR_HANDLE_MISMATCH` error.

**See Also**  
`libs cf(3LIB), scf_error(3SCF), scf_service_is_masked(3SCF), attributes(5)`
The `scf_handle_decode_fmri()` function decodes an FMRI string into a set of repository entries. Any number of the entity handles can be NULL. The validation and decoding of the FMRI are determined by the `flags` argument and by those arguments that are NULL.

If `flags` == 0, any FMRI is accepted as long as it is well-formed and exists in the repository.

If `SCF_DECODE_FMRI_EXACT` is set in `flags`, the last part of the FMRI must match the last non-null entity handle. For example, if property is NULL and `pg` is non-null, the FMRI must be a property group FMRI.

If `SCF_DECODE_FMRI_TRUNCATE` is set in `flags`, there is no check for the existence of any objects specified in the FMRI that follow the last non-null entity handle. For example, if property is NULL, `pg` is non-null, and a property FMRI is passed in, `scf_handle_decode_fmri()` succeeds as long as the property group exists, even if the referenced property does not exist.

If `SCF_DECODE_FMRI_REQUIRE_INSTANCE` (or `SCF_DECODE_FMRI_REQUIRE_NO_INSTANCE`) is set in `flags`, then the FMRI must (or must not) specify an instance.

If an error occurs, all of the entity handles that were passed to the function are reset.

The `scf_scope_to_fmri()`, `scf_service_to_fmri()`, `scf_instance_to_fmri()`, `scf_pg_to_fmri()`, and `scf_property_to_fmri()` functions convert an entity handle to an FMRI.

Upon successful completion, `scf_handle_decode_fmri()` returns 0. Otherwise, it returns -1.
Upon successful completion, `scf_scope_to_fmri()`, `scf_service_to_fmri()`, `scf_instance_to_fmri()`, `scf_pg_to_fmri()`, and `scf_property_to_fmri()` return the length of the FMRI. The buffer will be null-terminated if `sz` > 0, similar to `strlcpy(3C)`. Otherwise, they return -1 and the contents of buffer are undefined.

### Errors

The `scf_handle_decode_fmri()` function will fail if:

- **SCF_ERROR_BACKEND_ACCESS**  The storage mechanism that the repository server (`svc.configd(1M)`) chose for the operation denied access.
- **SCF_ERROR_CONNECTION_BROKEN**  The connection to the repository was lost.
- **SCF_ERROR_CONSTRAINT_VIOLATED**  The FMRI does not meet the restrictions requested in the flag argument.
- **SCF_ERROR_DELETED**  The object argument refers to an object that has been deleted.
- **SCF_ERROR_HANDLE_MISMATCH**  One or more of the entity handles was not derived from handle.
- **SCF_ERROR_INTERNAL**  An internal error occurred.
- **SCF_ERROR_INVALID_ARGUMENT**  The `fmri` argument is not a valid FMRI.
- **SCF_ERROR_NO_RESOURCES**  The server does not have adequate resources to complete the request.
- **SCF_ERROR_NOT_BOUND**  The handle is not currently bound.
- **SCF_ERROR_NOT_FOUND**  The FMRI is well-formed but there is no object in the repository matching it.
- **SCF_ERROR_NOT_SET**  Cannot use unset value.

The `scf_scope_to_fmri()`, `scf_service_to_fmri()`, `scf_instance_to_fmri()`, `scf_pg_to_fmri()`, and `scf_property_to_fmri()` functions will fail if:

- **SCF_ERROR_NOT_SET**  The `object` argument is not currently set.
- **SCF_ERROR_DELETED**  The object argument refers to an object that has been deleted.
- **SCF_ERROR_NOT_BOUND**  The handle is not currently bound.
- **SCF_ERROR_CONNECTION_BROKEN**  The connection to the repository was lost.

The `scf_error(3SCF)` function can be used to retrieve the error value.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libscf(3LIB), scf_error(3SCF), attributes(5)
**Name**
scf_instance_create, scf_instance_handle, scf_instance_destroy, scf_instance_get_parent,
scf_instance_get_name, scf_service_get_instance, scf_service_add_instance,
scf_instance_is_complete, scf_instance_delete – create and manipulate instance handles and
instances in the Service Configuration Facility

**Synopsis**
c { flag... } file... -lscf [ library... ]
#include <libscf.h>

```c
scf_instance_t *scf_instance_create(scf_handle_t *handle);
scf_handle_t *scf_instance_handle(scf_instance_t *inst);
void scf_instance_destroy(scf_instance_t *inst);
int scf_instance_get_parent(const scf_instance_t *inst,
    scf_service_t *svc);
ssize_t scf_instance_get_name(const scf_instance_t *inst,
    char *name, size_t size);
int scf_service_get_instance(const scf_service_t *svc,
    const char *name, scf_instance_t *inst);
int scf_service_add_instance(const scf_service_t *svc,
    const char *name, scf_instance_t *inst);
int scf_instance_is_complete(scf_instance_t *inst);
int scf_instance_delete(scf_instance_t *inst);
```

**Description**
Instances form the bottom layer of the Service Configuration Facility repository tree. An
instance is the child of a service and has two sets of children:

- **Property Groups**: These hold configuration information specific to this instance. See
  `scf_pg_create(3SCF), scf_iter_instance_pgs(3SCF), and scf_iter_instance_pgs_typed(3SCF).`

- **Snapshots**: These are complete configuration snapshots that hold unchanging
  copies of all of the property groups necessary to run the instance. See
  `scf_snapshot_create(3SCF) and scf_iter_instance_snapshots(3SCF).`

Not all instances retrieved by the functions described here are complete. Some instances may
contain only partial specifications, such as when customizations are delivered by a profile, but
the instance itself is not described in the manifest. The `scf_instance_is_complete()`
function can be used to determine whether the instance is complete.

See `smf(5)` for information about instances.

An `scf_instance_t` is an opaque handle that can be set to a single instance at any given time.
The `scf_instance_create()` function allocates and initializes a new `scf_instance_t` bound
to `handle`. The `scf_instance_destroy()` function destroys and frees `inst`. 
The `scf_instance_handle()` function retrieves the handle to which `inst` is bound.

The `scf_inst_get_parent()` function sets `svc` to the service that is the parent of `inst`.

The `scf_instance_get_name()` function retrieves the name of the instance to which `inst` is set.

The `scf_service_get_instance()` function sets `inst` to the child instance of the service `svc` specified by `name`.

The `scf_service_add_instance()` function sets `inst` to a new child instance of the service `svc` specified by `name`.

The `scf_instance_delete()` function deletes the instance to which `inst` is set, as well all of the children of the instance.

**Return Values**

Upon successful completion, `scf_instance_create()` returns a new `scf_instance_t`. Otherwise it returns `NULL`.

Upon successful completion, `scf_instance_handle()` returns the handle to which `inst` is bound. Otherwise, it returns `NULL`.

Upon successful completion, `scf_instance_get_name()` returns the length of the string written, not including the terminating null character. Otherwise it returns `-1`.

Upon successful completion, `scf_instance_get_parent()`, `scf_service_get_instance()`, `scf_service_add_instance()`, and `scf_instance_delete()` functions return `0`. Otherwise, they return `-1`.

The `scf_instance_is_complete()` function returns `1` if the instance is complete, `0` if it is not, and `-1` if there was an error.

**Errors**

The `scf_instance_create()` and `scf_instance_is_complete()` functions will fail if:

- `SCF_ERROR_HANDLE_DESTROYED`
  - An object was bound to a destroyed handle.

- `SCF_ERROR_INTERNAL`
  - An internal error occurred.

- `SCF_ERROR_INVALID_ARGUMENT`
  - The `handle` argument is `NULL`.

- `SCF_ERROR_NO_MEMORY`
  - There is not enough memory to allocate an `scf_instance_t`.

- `SCF_ERROR_NO_RESOURCES`
  - The server does not have adequate resources for a new instance handle.
The `scf_instance_handle()` function will fail if:

SCF_ERROR_HANDLE_DESTROYED

   The handle associated with `inst` has been destroyed.

The `scf_instance_get_name()`, `scf_instance_get_parent()`, `scf_instance_is_complete()`, and `scf_instance_delete()` functions will fail if:

SCF_ERROR_DELETED

   The instance has been deleted.

SCF_ERROR_NOT_SET

   The instance is not set.

SCF_ERROR_NOT_BOUND

   The repository handle is not bound.

SCF_ERROR_CONNECTION_BROKEN

   The connection to the repository was lost.

The `scf_service_add_instance()` function will fail if:

SCF_ERROR_EXISTS

   An instance named `name` already exists.

SCF_ERROR_INTERNAL

   An internal error occurred.

SCF_ERROR_NO_RESOURCES

   The server does not have the resources to complete the request.

SCF_ERROR_NOT_BOUND

   The handle is not bound.

The `scf_service_get_instance()` function will fail if:

SCF_ERROR_BACKEND_ACCESS

   The storage mechanism that the repository server (`svc.configd(1M)`) chose for the operation denied access.

SCF_ERROR_INTERNAL

   An internal error occurred.

SCF_ERROR_NOT_BOUND

   The handle is not bound.

SCF_ERROR_NOT_FOUND

   No instance specified by `name` was found.

SCF_ERROR_NO_RESOURCES

   The repository server is out of resources.

The `scf_service_add_instance()` and `scf_service_get_instance()` functions will fail if:
SCF_ERROR_NOT_SET
  The service is not set.

SCF_ERROR_DELETED
  The service has been deleted.

SCF_ERROR_INVALID_ARGUMENT
  The name argument is not a valid instance name.

SCF_ERROR_HANDLE_MISMATCH
  The service and instance are not derived from the same handle.

SCF_ERROR_CONNECTION_BROKEN
  The connection to the repository was lost.

The scf_instance_get_parent() function will fail if:

SCF_ERROR_HANDLE_MISMATCH
  The service and instance arguments are not derived from the same handle.

The scf_service_add_instance() and scf_instance_delete() functions will fail if:

SCF_ERROR_PERMISSION_DENIED
  The user does not have sufficient privileges to create or delete an instance.

SCF_ERROR_BACKEND_READONLY
  The repository backend is read-only.

SCF_ERROR_BACKEND_ACCESS
  The repository backend refused the modification.

The scf_instance_delete() function will fail if:

SCF_ERROR_NO_RESOURCES
  The server does not have adequate resources for a new instance handle.

The scf_instance_is_complete() function will fail if:

SCF_ERROR_BACKEND_ACCESS
  The storage mechanism that the repository server (svc.configd(1M)) chose for the operation denied access.

SCF_ERROR_NOT_FOUND
  The complete property or one of its parents was not found.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
### ATTRIBUTE VALUE

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `svc.configd(IM)`, `libscf(LIB)`, `scf_error(SCF)`, `scf_iter_instance_pgs(SCF)`, `scf_iter_instance_pgs_typed(SCF)`, `scf_iter_instance_snapshots(SCF)`, `scf_pg_create(SCF)`, `scf_snapshot_create(SCF)`, `attributes(5)`, `smf(5)`

**Notes**

Instance names are of the form:

```
[domain,]identifier
```

where `domain` is either a stock ticker symbol such as ORCL or a Java-style reversed domain name such as `com.oracle`. Identifiers begin with a letter or underscore and contain only letters, digits, underscores, and dashes.
**Name**
scf_iter_create, scf_iter_handle, scf_iter_destroy, scf_iter_reset, scf_iter_handle_scopes, scf_iter_scope_services, scf_iter_service_instances, scf_iter_service_pgs, scf_iter_service_pgs_TYPED, scf_iter_instance_snapshots, scf_iter_snaplevel_pgs, scf_iter_snaplevel_pgs_TYPED, scf_iter_instance_pgs, scf_iter_instance_pgs_TYPED, scf_iter_instance_pgs_composed, scf_iter_instance_pgs_TYPED_composed, scf_iter_pg_properties, scf_iter_property_values, scf_iter_next_scope, scf_iter_next_service, scf_iter_next_instance, scf_iter_next_snapshot, scf_iter_next_pg, scf_iter_next_property, scf_iter_next_value – iterate through the Service Configuration Facility repository

**Synopsis**

```
cc [ flag... ] file... -lscf [ library... ]
#include <libscf.h>

scf_iter_t *scf_iter_create(scf_handle_t *handle);
scf_handle_t *scf_iter_handle(scf_iter_t *iter);
void scf_iter_destroy(scf_iter_t *iter);
void scf_iter_reset(scf_iter_t *iter);
int scf_iter_handle_scopes(scf_iter_t *iter, const scf_handle_t *h);
int scf_iter_scope_services(scf_iter_t *iter, const scf_scope_t *parent);
int scf_iter_service_instances(scf_iter_t *iter, const scf_service_t *parent);
int scf_iter_service_pgs(scf_iter_t *iter, const scf_service_t *parent);
int scf_iter_service_pgs_TYPED(scf_iter_t *iter, const scf_service_t *parent, const char *pgtype);
int scf_iter_instance_snapshots(scf_iter_t *iter, const scf_instance_t *parent);
int scf_iter_snaplevel_pgs(scf_iter_t *iter, const scf_snaplevel_t *parent);
int scf_iter_snaplevel_pgs_TYPED(scf_iter_t *iter, const scf_snaplevel_t *parent, const char *pgtype);
int scf_iter_instance_pgs(scf_iter_t *iter, const scf_instance_t *parent);
int scf_iter_instance_pgs_TYPED(scf_iter_t *iter, const scf_instance_t *parent, const char *pgtype);
int scf_iter_instance_pgs_composed(scf_iter_t *iter, const scf_instance_t *instance, const scf_snapshot_t *snapshot);
int scf_iter_instance_pgs_TYPED_composed(scf_iter_t *iter, const scf_instance_t *instance, const scf_snapshot_t *snapshot, const char *pgtype);
int scf_iter_pg_properties(scf_iter_t *iter, const scf_propertygroup_t *parent);
```
The scf_iter_create() function creates a new iterator associated with handle. The scf_iter_destroy() function destroys an iteration.

The scf_iter_reset() function releases any resources involved with an active iteration and returns the iterator to its initial state.

The scf_iter_handle_scopes(), scf_iter_scope_services(), scf_iter_service_instances(), scf_iter_instance_snapshots(), scf_iter_service_pgs(), scf_iter_instance_pgs(), scf_iter_snaplevel_pgs(), scf_iter_pg_properties(), and scf_iter_property_values() functions set up a new iteration of all the children parent of a particular type. The scf_iter_property_values() function will iterate over values in the order in which they were specified with scf_entry_add_value(3SCF).

The scf_iter_service_pgs_typed(), scf_iter_instance_pgs_typed(), and scf_iter_snaplevel_pgs_typed() functions iterate over the child property groups of parent, but restrict them to a particular property group type.

The scf_iter_instance_pgs_composed() function sets up a new iteration of the composed view of instance's children at the time snapshot was taken. If snapshot is NULL, the current properties are used. The composed view of an instance's properties is the union of the properties of the instance and its ancestors. Properties of the instance take precedence over properties of the service with the same name, including property group name. Property groups retrieved with this iterator might not have instance as their parent and properties retrieved from such property groups might not have the indicated property group as their parent. If instance and its parent have property groups with the same name but different types, the properties in the property group of the parent are excluded. The scf_iter_instance_pgs_composed() function behaves as scf_iter_instance_pgs_composed(), except the property groups of the type pgttype are returned.

The scf_iter_next_scope(), scf_iter_next_service(), scf_iter_next_instance(), scf_iter_next_snapshot(), scf_iter_next_pg(), scf_iter_next_property(), and scf_iter_next_value() functions retrieve the next element of the iteration.
Upon successful completion, `scf_iter_create()` returns a pointer to a new iterator. Otherwise, it returns NULL.

Upon successful completion, `scf_iter_handle()` returns the handle associated with `iter`. Otherwise, it returns NULL.

Upon successful completion, `scf_iter_handle_scopes()`, `scf_iter_scope_services()`, `scf_iter_service_instances()`, `scf_iter_instance_snapshots()`, `scf_iter_service_pgs()`, `scf_iter_instance_pgs()`, `scf_iter_snaplevel_pgs()`, `scf_iter_pg_properties()`, `scf_iter_property_values()`, `scf_iter_service_pgs_typed()`, `scf_iter_instance_pgs_composed()`, `scf_iter_instance_pgs_typed()`, `scf_iter_instance_pgs_typed_composed()`, and `scf_iter_snaplevel_pgs_typed()` return 0. Otherwise, they return -1.

Upon successful completion, `scf_iter_next_scope()`, `scf_iter_next_service()`, `scf_iter_next_instance()`, `scf_iter_next_snapshot()`, `scf_iter_next_pg()`, `scf_iter_next_property()`, and `scf_iter_next_value()` return 1. If the iterator is complete, they return 0. Otherwise, they return -1.

Errors
The `scf_iter_create()` function will fail if:
- `SCF_ERROR_INVALID_ARGUMENT` The handle argument is NULL.
- `SCF_ERROR_NO_MEMORY` There is no memory available.
- `SCF_ERROR_NO_RESOURCES` The server does not have adequate resources for a new iteration.

The `scf_iter_handle()` function will fail if:
- `SCF_ERROR_HANDLE.Destroyed` The handle associated with `iter` has been destroyed.

The `scf_iter_next_value()` function will fail if:
- `SCF_ERROR_PERMISSION_DENIED` The value could not be read due to access restrictions.

The `scf_iter_handle.scopes()`, `scf_iter_scope_services()`, `scf_iter_service_instances()`, `scf_iter_instance_snapshots()`, `scf_iter_service_pgs()`, `scf_iter_instance_pgs()`, `scf_iter_snaplevel_pgs()`, `scf_iter_pg_properties()`, `scf_iter_property_values()`, `scf_iter_service_pgs_typed()`, `scf_iter_instance_pgs_composed()`, `scf_iter_instance_pgs_typed()`, `scf_iter_instance_pgs_typed_composed()`, and `scf_iter_snaplevel_pgs_typed()` functions will fail if:
- `SCF_ERROR_DELETED` The parent has been deleted.
- `SCF_ERROR_NOT_SET` The parent is not set.
- `SCF_ERROR_NOT_BOUND` The handle is not bound.
The connection to the repository was lost.

The iter and parent arguments are not derived from the same handle.

The scf_iter_service_pgs_typed(), scf_iter_instance_pgs_typed(),
scf_iter_instance_pgs_typed_composed(), and scf_iter_snaplevel_pgs_typed()
functions will fail if:

The pgtype argument is not a valid property group type.

The scf_iter_next_service(), scf_iter_next_instance(), scf_iter_next_snapshot(),
scf_iter_next_pg(), scf_iter_next_property(), and scf_iter_next_value() functions
will fail if:

The parent the iterator is attached to has been deleted.

The scf_iter_next_scope(), scf_iter_next_service(), scf_iter_next_instance(),
scf_iter_next_snapshot(), scf_iter_next_pg(), scf_iter_next_property(), and
scf_iter_next_value() functions will fail if:

The iterator is not set.

The requested object type does not match the type the
iterator is walking.

The handle is not bound.

The iter and parent arguments are not derived from the
same handle.

The connection to the repository was lost.

The server does not have the resources to complete the
request.

The scf_error(3SCF) function can be used to retrieve the error value.
Examples

**EXAMPLE 1**  Iterate over all instances under a service.

```c
scf_iter_t *iter = scf_iter_create(handle);

if (iter == NULL || scf_iter_service_instances(iter, parent) == -1) {
    /* failure */
}
while ((r = scf_iter_next_instance(iter, child)) > 0) {
    /* process child */
}
if (r < 0) {
    /* failure */
}
scf_iter_destroy(iter);
```

**EXAMPLE 2**  Connect to the repository, walk all services and instances and print their FMRI's.

```c
scf_handle_t *handle = scf_handle_create(SCF_VERSION);
scf_scope_t *scope = scf_scope_create(handle);
scf_service_t *svc = scf_service_create(handle);
scf_instance_t *inst = scf_instance_create(handle);
scf_iter_t *svc_iter = scf_iter_create(handle);
scf_iter_t *inst_iter = scf_iter_create(handle);
size_t sz = scf_limit(SCF_LIMIT_MAX_FMRI_LENGTH) + 1;
char *fmri = malloc(sz + 1);

int r;

if (handle == NULL || scope == NULL || svc == NULL ||
    inst == NULL || svc_iter == NULL || inst_iter == NULL ||
    fmri == NULL) {
    /* failure */
}
if (scf_handle_bind(handle) == -1 ||
    scf_handle_get_scope(handle, SCF_SCOPE_LOCAL, scope) == -1 ||
    scf_iter_scope_services(svc_iter, scope) == -1) {
    /* failure */
}
while ((r = scf_iter_next_service(svc_iter, svc)) > 0) {
    if (scf_service_to_fmri(svc, fmri, sz) < 0) {
        /* failure */
    }
    puts(fmri);
    if (scf_iter_service_instances(inst_iter, svc) < 0) {
        /* failure */
    }
    while ((r = scf_iter_next_instance(inst_iter, inst)) > 0) {
        /* process inst */
    }
```
EXAMPLE 2  Connect to the repository, walk all services and instances and print their FMRIs.

(Continued)

if (scf_instance_to_fmri(inst, fmri, sz) < 0) {
    /* failure */
}
puts(fmri);
}
if (r < 0)
break;
}
if (r < 0) {
    /* failure */
}

scf_handle_destroy(handle);
scf_scope_destroy(scope);
scf_service_destroy(svc);
scf_instance_destroy(inst);
scf_iter_destroy(svc_iter);
scf_iter_destroy(inst_iter);

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libscf(3LIB), scf_entry_add_value(3SCF), scf_error(3SCF), scf_handle_create(3SCF), attributes(5)
Name scf_limit – limit information for Service Configuration Facility

Synopsis cc [ flag... ] file... -lscf [ library... ]
#include <libscf.h>

ssize_t scf_limit(uint32_t name);

Description The scf_limit() function returns information about implementation-defined limits in the service configuration facility. These limits are generally maximum lengths for various strings. The values returned do not change during the execution of a program, but they should not be cached between executions.

The available values for name are:

- **SCF_LIMIT_MAX_FMRI_LENGTH**
  Return the maximum length of an FMRI the service configuration facility accepts.

- **SCF_LIMIT_MAX_PG_TYPE_LENGTH**
  Return the maximum length for property group types in the service configuration facility.

- **SCF_LIMIT_MAX_NAME_LENGTH**
  Return the maximum length for names in the service configuration facility. This value does not include space for the required terminating null byte.

- **SCF_LIMIT_MAX_VALUE_LENGTH**
  Return the maximum string length a scf_value_t can hold, not including the terminating null byte.

Lengths do not include space for the required terminating null byte.

Return Values Upon successful completion, scf_limit() returns the requested value. Otherwise, it returns -1.

Errors The scf_limit() function will fail if:

- **SCF_ERROR_INVALID_ARGUMENT**
  The name argument is not a recognized request.

The scf_error(3SCF) function can be used to retrieve the error value.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also libscf(3LIB), scf_error(3SCF), attributes(5)
### Name

scf_pg_create, scf_pg_handle, scf_pg_destroy, scf_pg_get_parent_service, scf_pg_get_parent_instance, scf_pg_get_parent_snaplevel, scf_pg_get_name, scf_pg_get_type, scf_pg_get_flags, scf_pg_update, scf_service_get_pg, scf_service_add_pg, scf_instance_get_pg, scf_instance_get_pg_composed, scf_instance_add_pg, scf_snaplevel_get_pg, scf_pg_delete, scf_pg_get_underlying_pg – create and manipulate property group handles and property groups in the Service Configuration Facility

### Synopsis

```c
cc [ flag... ] file... -lscf [ library... ]
#include <libscf.h>

scf_propertygroup_t *scf_pg_create(scf_handle_t *handle);
scf_handle_t *scf_pg_handle(scf_propertygroup_t *pg);
void scf_pg_destroy(scf_propertygroup_t *pg);
int scf_pg_get_parent_service(const scf_propertygroup_t *pg, scf_service_t *svc);
int scf_pg_get_parent_instance(const scf_propertygroup_t *pg, scf_instance_t *inst);
int scf_pg_get_parent_snaplevel(const scf_propertygroup_t *pg, scf_snaplevel_t *level);
ssize_t scf_pg_get_name(const scf_propertygroup_t *pg, char *buf, size_t size);
ssize_t scf_pg_get_type(const scf_propertygroup_t *pg, char *buf, size_t size);
int scf_pg_get_flags(const scf_propertygroup_t *pg, uint32_t *out);
int scf_pg_update(const scf_propertygroup_t *pg);
int scf_service_get_pg(const scf_service_t *svc, const char *name, scf_propertygroup_t *pg);
int scf_service_add_pg(const scf_service_t *svc, const char *name, const char *group_type, uint32_t flags, scf_propertygroup_t *pg);
int scf_instance_get_pg(const scf_instance_t *inst, const char *name, scf_propertygroup_t *pg);
int scf_instance_get_pg_composed(const scf_instance_t *inst, const scf_snapshot_t *snapshot, const char *name, scf_propertygroup_t *pg);
int scf_instance_add_pg(const scf_instance_t *inst, const char *name, const char *group_type, uint32_t flags, scf_propertygroup_t *pg);
int scf_snaplevel_get_pg(const scf_snaplevel_t *level, const char *name, scf_propertygroup_t *pg);
```
int scf_pg_delete(scf_propertygroup_t *pg);
int scf_pg_get_underlying_pg(const scf_propertygroup_t *pg,
    scf_propertygroup_t *our);

Description

Property groups are an atomically-updated group of typed properties. Property groups of
services (see scf_service_create(3SCF)) or instances (see scf_instance_create(3SCF))
are modifiable. Property groups of snaplevels (see scf_snaplevel_create(3SCF)) are not
modifiable.

An scf_propertygroup_t is an opaque handle that can be set to a single property group at
any given time. When an scf_propertygroup_t is set, it references a frozen-in-time version
of the property group to which it is set. Updates to the property group will not be visible until
either scf_pg_update() is called or the property group is set again.

This static view is propagated to the scf_property Ts set to children of the property group.
They will not see updates, even if the scf_propertygroup_t is updated.

The scf_pg_create() function allocates and initializes a new scf_propertygroup_t bound
to handle. The scf_pg_destroy() function destroys and frees pg.

The scf_pg_handle() function retrieves the handle to which pg is bound.

The scf_pg_get_parent_service(), scf_pg_get_parent_instance(), and
scf_pg_get_parent_snaplevel() functions retrieve the property group's parent, if it is of the
requested type.

The scf_pg_get_name() and scf_pg_get_type() functions retrieve the name and type,
respectively, of the property group to which pg is set.

The scf_pg_get_flags() function retrieves the flags for the property group to which pg is set.
If SCF_PG_FLAG_NONPERSISTENT is set, the property group is not included in snapshots and
will lose its contents upon system shutdown or reboot. Non-persistent property groups are
mainly used for smf-internal state. See smf(5).

The scf_pg_update() function ensures that pg is attached to the most recent version of the pg
to which it is set.

The scf_service_get_pg(), scf_instance_get_pg(), and scf_snaplevel_get_pg() functions set pg to the property group specified by name in the service specified by svc, the
instance specified by inst, or the snaplevel specified by level, respectively.

The scf_instance_get_pg_composed() function sets pg to the property group specified by
name in the composed view of inst at the time snapshot was taken. If snapshot is NULL, the
current properties are used. The composed view of an instance's properties is the union of the
properties of the instance and its ancestors. Properties of the instance take precedence over
properties of the service with the same name (including the property group name). After a
successful call to scf_instance_get_pg_composed(), the parent of pg might not be inst, and
the parents of properties obtained from `pg` might not be `pg`. If `inst` and its parent have property groups with the same name but different types, the properties in the property group of the parent are excluded.

The `scf_service_add_pg()` and `scf_instance_add_pg()` functions create a new property group specified by `name` whose type is `group_type`, and attach the `pg` handle (if non-null) to the new object. The `flags` argument must be either 0 or `SCF_PG_FLAG_NONPERSISTENT`.

The `scf_pg_delete()` function deletes the property group. Versions of the property group in snapshots are not affected.

The `scf_pg_get_underlying_pg()` function gets the first existing underlying property group. If the property group specified by `pg` is an instance property group, `out` is set to the property group of the same name in the instance’s parent.

Applications can use a transaction to modify a property group. See `scf_transaction_create(3SCF)`.

**Return Values**

Upon successful completion, `scf_pg_create()` returns a new `scf_propertygroup_t`. Otherwise, it returns NULL.

Upon successful completion, `scf_pg_handle()` returns a pointer to the handle to which `pg` is bound. Otherwise, it returns NULL.

Upon successful completion, `scf_instance_handle()` returns the handle instance with which it is associated. Otherwise, it returns NULL.

Upon successful completion, `scf_pg_get_name()` and `scf_pg_get_type()` return the length of the string written, not including the terminating null byte. Otherwise, they return -1.

The `scf_pg_update()` function returns 1 if the object was updated, 0 if the object was already up to date, and -1 on failure.

Upon successful completion, `scf_pg_get_parent_service()`, `scf_pg_get_parent_snaplevel()`, `scf_pg_get_flags()`, `scf_service_get_pg()`, `scf_service_add_pg()`, `scf_pg_get_parent_instance()`, `scf_instance_get_pg()`, `scf_instance_get_pg_composed()`, `scf_instance_add_pg()`, `scf_snaplevel_get_pg()`, `scf_pg_delete()`, and `scf_pg_get_underlying_pg()` return 0. Otherwise, they return -1.

**Errors**

The `scf_pg_create()` function will fail if:

- `SCF_ERROR_INVALID_ARGUMENT`
  - The `handle` argument is NULL.

- `SCF_ERROR_NO_MEMORY`
  - There is not enough memory to allocate an `scf_propertygroup_t`.

- `SCF_ERROR_NO_RESOURCES`
  - The server does not have adequate resources for a new property group handle.
The `scf_pg_handle()` function will fail if:

**SCF_ERROR_HANDLE_DESTROYED**
The handle associated with `pg` has been destroyed.

The `scf_pg_update()` function will fail if:

**SCF_ERROR_CONNECTION_BROKEN**
The connection to the repository was lost.

**SCF_ERROR_DELETED**
An ancestor of the property group specified by `pg` has been deleted.

**SCF_ERROR_INTERNAL**
An internal error occurred. This can happen if `pg` has been corrupted.

**SCF_ERROR_INVALID_ARGUMENT**
The `pg` argument refers to an invalid `scf_propertygroup_t`.

**SCF_ERROR_NOT_BOUND**
The handle is not bound.

**SCF_ERROR_NOT_SET**
The property group specified by `pg` is not set.

The `scf_service_get_pg()`, `scf_instance_get_pg()`, `scf_instance_get_pg_composed()`, `scf_snaplevel_get_pg()`, and `scf_pg_get_underlying_pg()` functions will fail if:

**SCF_ERROR_BACKEND_ACCESS**
The storage mechanism that the repository server (`svc.configd(1M)`) chose for the operation denied access.

**SCF_ERROR_INTERNAL**
An internal error occurred.

**SCF_ERROR_NO_RESOURCES**
The server does not have the resources to complete the request.

The `scf_pg_get_name()`, `scf_pg_get_type()`, `scf_pg_get_flags()`, `scf_pg_get_parent_service()`, `scf_pg_get_parent_snaplevel()`, and `scf_pg_get_parent_instance()` functions will fail if:

**SCF_ERROR_DELETED**
The property group specified by `pg` has been deleted.

**SCF_ERROR_NOT_SET**
The property group specified by `pg` is not set.

**SCF_ERROR_NOT_BOUND**
The handle is not bound.
SCF_ERROR_CONNECTION_BROKEN
    The connection to the repository was lost.

The scf_pg_get_parent_service(), scf_pg_get_parent_snaplevel(), and
scf_pg_get_parent_instance() functions will fail if:

SCF_ERROR_CONSTRAINT_VIOLATED
    The requested parent type does not match the actual type of the parent of the property
group specified by pg.

SCF_ERROR_HANDLE_MISMATCH
    The property group and either the instance, the service, or the snaplevel are not derived
from the same handle.

The scf_instance_get_pg(), scf_instance_get_pg_composed(), scf_service_get_pg(),
scf_pg_get_underlying_pg(), and scf_snaplevel_get_pg() functions will fail if:

SCF_ERROR_NOT_FOUND
    The property group specified by name was not found.

The scf_service_add_pg(), scf_service_get_pg(), scf_instance_add_pg(),
scf_instance_get_pg(), scf_instance_get_pg_composed(), and
scf_snaplevel_get_pg() functions will fail if:

SCF_ERROR_DELETED
    The service or instance has been deleted.

SCF_ERROR_NOT_SET
    The instance is not set.

SCF_ERROR_INVALID_ARGUMENT
    The value of the name argument is not a valid property group name.

SCF_ERROR_HANDLE_MISMATCH
    The property group and either the instance, the service, or the level are not derived from
the same handle.

SCF_ERROR_NOT_BOUND
    The handle is not bound.

SCF_ERROR_CONNECTION_BROKEN
    The connection to the repository was lost.

The scf_service_add_pg() and scf_instance_add_pg() functions will fail if:

SCF_ERROR_PERMISSION_DENIED
    The caller does not have permission to create the requested property group.

SCF_ERROR_BACKEND_READONLY
    The repository backend is read-only.
Therepositorybackendrefusedthemodification.

A {service,instance,property group} named name already exists.

The server does not have the resources to complete the request.

The scf_pg_delete() function will fail if:

The repository backend refused the modification.

The repository backend is read-only.

The connection to the repository was lost.

The property group has been deleted by someone else.

The server does not have adequate resources for a new property group handle.

The property group has not been set.

The caller does not have permission to delete this property group.

The scf_pg_get_underlying_pg() function will fail if:

The connection to the repository was lost.

A required constraint was not met.

The property group has been deleted.

The property group and out are not derived from the same handle.

An argument is invalid.

The handle is not bound.
SCF_ERROR_NOT_SET
The property group has not been set.

The scf_error(3SCF) function can be used to retrieve the error value.

**Examples**

**EXAMPLE 1** Perform a layered lookup of `name` in `pg`.

```c
int layered_lookup(scf_propertygroup_t *pg, const char *name,
                   scf_property_t *out) {
  scf_handle_t *handle = scf_pg_handle(out);
  scf_propertygroup_t *new_pg;
  scf_propertygroup_t *cur, *other;
  int state = 0;

  if (handle == NULL) {
    return (-1);
  }
  new_pg = scf_pg_create(handle);
  if (new_pg == NULL) {
    return (-1);
  }
  for (;;) {
    cur = state ? pg : new_pg;
    other = state ? new_pg : pg;
    state = !state;

    if (scf_pg_get_property(cur, name, out) != -1) {
      scf_pg_destroy(new_pg);
      return (SUCCESS);
    }
    if (scf_pg_get_underlying_pg(cur, other) == -1)
      break;
  }
  scf_pg_destroy(new_pg);
  return (NOT_FOUND);
}
```

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also** libscf(3LIB), scf_error(3SCF), scf_handle_decode_fmri(3SCF),
scf_instance_create(3SCF), scf_pg_to_fmri(3SCF), scf_service_create(3SCF),
scf_snaplevel_create(3SCF), scf_transaction_create(3SCF), attributes(5), smf(5)
scf_property_create(3SCF)

**Name**
scf_property_create, scf_property_handle, scf_property_destroy, scf_property_get_name, scf_property_type, scf_property_is_type, scf_type_to_string, scf_string_to_type, scf_property_get_value, scf_pg_get_property – create and manipulate property handles in the Service Configuration Facility

**Synopsis**
cc [ flag... ] file... -lscc [ library... ]
#include <libscf.h>

```c
scf_property_t *scf_property_create(scf_handle_t *handle);
scf_handle_t *scf_property_handle(scf_property_t *prop);
void scf_property_destroy(scf_property_t *prop);
ssize_t scf_property_get_name(const scf_property_t *prop,
    char *buf, size_t size);
int scf_property_type(const scf_property_t *prop,
    scf_type_t *type);
int scf_property_is_type(const scf_property_t *prop,
    scf_type_t type);
const char *scf_type_to_string(scf_type_t type);
scf_type_t scf_string_to_type(const char *type);
int scf_property_get_value(const scf_property_t *prop,
    scf_value_t *value);
int scf_pg_get_property(const scf_propertygroup_t *pg,
    const char *name, scf_property_t *prop);
```

**Description**
Properties are named sets of values of one type. They are grouped into property groups (see scf_pg_create(3SCF)) that are updated atomically using transactions (see scf_transaction_create(3SCF)).

An scf_property_t is an opaque handle that can be set to a single property at any given time. When set, it inherits the point-in-time from the source scf_propertygroup_t and does not change until reset.

The scf_property_create() function allocates and initializes a new scf_property_t bound to handle. The scf_property_destroy() function destroys and frees prop.

The scf_property_handle() function returns the handle to which prop is bound.

The scf_property_type() function retrieves the type of the property to which prop is set.

The scf_property_is_type() function determines if the property is compatible with type. See scf_value_create(3SCF).

The scf_type_to_string() function returns the string name of the type supplied. If the type is invalid or unknown, it returns “unknown”.

```c
Properties are named sets of values of one type. They are grouped into property groups (see scf_pg_create(3SCF)) that are updated atomically using transactions (see scf_transaction_create(3SCF)).

An scf_property_t is an opaque handle that can be set to a single property at any given time. When set, it inherits the point-in-time from the source scf_propertygroup_t and does not change until reset.

The scf_property_create() function allocates and initializes a new scf_property_t bound to handle. The scf_property_destroy() function destroys and frees prop.

The scf_property_handle() function returns the handle to which prop is bound.

The scf_property_type() function retrieves the type of the property to which prop is set.

The scf_property_is_type() function determines if the property is compatible with type. See scf_value_create(3SCF).

The scf_type_to_string() function returns the string name of the type supplied. If the type is invalid or unknown, it returns “unknown”.
```
The `scf_string_to_type()` function returns the `scf_type_t` definition of the string supplied. If the string does not translate to an existing type, it returns `SCF_TYPE_INVALID`.

The `scf_property_get_value()` function retrieves the single value that the property to which `prop` is set contains. If the property has more than one value, the `value` argument is set to one of the values. To retrieve all values associated with a property, see `scf_iter_property_values(3SCF)`.

The `scf_pg_get_property()` function sets `prop` to the property specified by `name` in the property group specified by `pg`.

**Return Values**

Upon successful completion, `scf_property_create()` returns a new `scf_property_t`. Otherwise, it returns `NULL`.

Upon successful completion, `scf_property_get_name()` function returns the length of the string written, not including the terminating null byte. Otherwise, it returns `-1`.

Upon successful completion, `scf_pg_get_property()`, `scf_property_type()`, `scf_property_is_type()`, and `scf_property_get_value()` functions return `0`. Otherwise, they return `-1`.

Upon successful completion, `scf_type_to_string()` returns a string of the type supplied.

Upon successful completion, `scf_string_to_type()` returns the `scf_type_t` definition of the string supplied.

**Errors**

The `scf_property_create()` function will fail if:

- `SCF_ERROR_INVALID_ARGUMENT` The value of the `handle` argument is `NULL`.
- `SCF_ERROR_NO_MEMORY` There is not enough memory to allocate an `scf_property_t`.
- `SCF_ERROR_NO_RESOURCES` The server does not have adequate resources for a new property handle.

The `scf_property_handle()` function will fail if:

- `SCF_ERROR_HANDLE DESTROYED` The handle associated with `prop` has been destroyed.

The `scf_property_get_name()`, `scf_property_type()`, `scf_property_is_type()`, and `scf_property_get_value()` functions will fail if:

- `SCF_ERROR_CONNECTION BROKEN` The connection to the repository was lost.
- `SCF_ERROR_DELETED` The property’s parent property group or an ancestor has been deleted.
- `SCF_ERROR_NOT_BOUND` The handle was never bound or has been unbound.
The property is not set.
The scf_property_is_type() function will fail if:

SCF_ERROR_INVALID_ARGUMENT The type argument is not a valid type.

SCF_ERROR_TYPE_MISMATCH The prop argument is not of a type compatible with type.

The scf_pg_get_property() function will fail if:

SCF_ERROR_BACKEND_ACCESS The storage mechanism that the repository server (svc.configd(1M)) chose for the operation denied access.

SCF_ERROR_CONNECTION_BROKEN The connection to the repository was lost.

SCF_ERROR_DELETED The property group or an ancestor has been deleted.

SCF_ERROR_HANDLE_MISMATCH The property group and property are not derived from the same handle.

SCF_ERROR_INTERNAL An internal error occurred.

SCF_ERROR_INVALID_ARGUMENT The value of the name argument is not a valid property name.

SCF_ERROR_NO_RESOURCES The server does not have the resources to complete the request.

SCF_ERROR_NOT_BOUND The handle was never bound or has been unbound.

SCF_ERROR_NOT_FOUND The property specified by name was not found.

SCF_ERROR_NOT_SET The property group specified by pg is not set.

The scf_property_get_value() function will fail if:

SCF_ERROR_CONSTRAINT_VIOLATED The property has more than one value associated with it. The value argument will be set to one of the values.

SCF_ERROR_HANDLE_MISMATCH The property and value are derived from different handles.

SCF_ERROR_NOT_FOUND The property has no values associated with it. The value argument will be reset.

SCF_ERROR_PERMISSION_DENIED The value could not be read due to access restrictions.

The scf_error(3SCF) function can be used to retrieve the error value.

**Attributes**  See attributes(5) for descriptions of the following attributes:
scf_property_create(3SCF)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libscf(3LIB), scf_error(3SCF), scf_handle_decode_fmri(3SCF), scf_iter_property_values(3SCF), scf_pg_create(3SCF), scf_property_to_fmri(3SCF), scf_transaction_create(3SCF), scf_value_create(3SCF), attributes(5)
**Name**
sclf\_scope\_create, scf\_scope\_handle, scf\_scope\_destroy, scf\_scope\_get\_name, scf\_handle\_get\_scope – create and manipulate scope handles in the Service Configuration Facility

**Synopsis**
```
cc [ flag... ] file... -lsclf [ library... ]
#include <libscf.h>

scf\_scope\_t *scf\_scope\_create(scf\_handle\_t *handle);
scf\_handle\_t *scf\_scope\_handle(scf\_scope\_t *sc);
void scf\_scope\_destroy(scf\_scope\_t *sc);
ssize_t scf\_scope\_get\_name(scf\_scope\_t *sc, char *buf, size\_t size);
int scf\_handle\_get\_scope(scf\_handle\_t *handle, const char *name, scf\_scope\_t *out);
```

**Description**
Scopes are the top level of the Service Configuration Facility's repository tree. The children of a scope are services (see scf\_service\_create(3SCF)) and can be walked using scf\_iter\_scope\_services(3SCF).

There is a distinguished scope with the name SCF\_SCOPE\_LOCAL that is the root for all available services on the local machine. In the current implementation, there are no other scopes.

An scf\_scope\_t is an opaque handle that can be set to a single scope at any given time. The scf\_scope\_create() function allocates a new scf\_scope\_t bound to handle. The scf\_scope\_destroy() function destroys and frees sc.

The scf\_scope\_handle() function retrieves the handle to which sc is bound.

The scf\_scope\_get\_name() function retrieves the name of the scope to which sc is set.

The scf\_handle\_get\_scope() function sets out to the scope specified by name for the repository handle specified by handle. The scf\_iter\_handle\_scopes(3SCF) and scf\_iter\_next\_scope(3SCF) calls can be used to iterate through all available scopes.

**Return Values**
Upon successful completion, scf\_scope\_create() returns a new scf\_scope\_t. Otherwise, it returns NULL.

Upon successful completion, scf\_scope\_handle() returns the handle to which sc is bound. Otherwise, it returns NULL.

Upon successful completion, scf\_scope\_get\_name() returns the length of the string written, not including the terminating null byte. Otherwise, it returns -1.

Upon successful completion, scf\_handle\_get\_scope() returns 0. Otherwise, it returns -1.
Errors  The scf_scope_create() function will fail if:

```
SCF_ERROR_INVALID_ARGUMENT       The value of the handle argument is NULL.
SCF_ERROR_NO_MEMORY             There is not enough memory to allocate an scf_scope_t.
SCF_ERROR_NO_RESOURCES          The server does not have adequate resources for a new scope handle.
```

The scf_scope_handle() function will fail if:

```
SCF_ERROR_HANDLE_DESTROYED      The handle associated with sc has been destroyed.
```

The scf_scope_get_name() function will fail if:

```
SCF_ERROR_NOT_SET               The scope is not set.
SCF_ERROR_NOT_BOUND             The handle is not bound.
SCF_ERROR_CONNECTION_BROKEN     The connection to the repository was lost.
```

The scf_handle_get_scope() function will fail if:

```
SCF_ERROR_NOT_FOUND            No scope named name was found.
SCF_ERROR_INVALID_ARGUMENT     The name argument is not a valid scope name.
SCF_ERROR_NOT_BOUND            The handle is not bound.
SCF_ERROR_CONNECTION_BROKEN    The connection to the repository was lost.
SCF_ERROR_HANDLE_MISMATCH     The value of the out argument is not derived from handle.
```

The scf_error(3SCF) function can be used to retrieve the error value.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libscf(3LIB), scf_error(3SCF), scf_handle_decode_fmri(3SCF),
scf_iter_handle_scopes(3SCF), scf_iter_next_scope(3SCF),
scf_iter_scope_services(3SCF), scf_scope_to_fmri(3SCF),
scf_service_create(3SCF), attributes(5)
Name

scf_service_create, scf_service_handle, scf_service_destroy, scf_service_get_parent,
scf_service_get_name, scf_scope_get_service, scf_scope_add_service, scf_service_delete –
create and manipulate service handles and services in the Service Configuration Facility

Synopsis

cc [ flag... ] file... -lsclf [ library... ]
#include <libscf.h>

scf_service_t *scf_service_create(scf_handle_t *handle);
scf_handle_t *scf_service_handle(scf_service_t *svc);
void scf_service_destroy(scf_service_t *svc);
int scf_service_get_parent(scf_service_t *svc, scf_scope_t *sc);
ssize_t scf_service_get_name(const scf_service_t *svc, char *buf,
size_t size);
int scf_scope_get_service(const scf_scope_t *sc, const char *name,
scf_service_t *svc);
int scf_scope_add_service(const scf_scope_t *sc, const char *name,
scf_service_t *svc);
int scf_service_delete(scf_service_t *svc);

Description

Services form the middle layer of the Service Configuration Facility repository tree. Services
are children of a scope (see scf_scope_create(3SCF)) and have three sets of children:

Property groups These hold configuration information shared by all of the instances of the
service. See scf_pg_create(3SCF), scf_iter_service_pgs(3SCF), and
scf_iter_service_pgs_typed(3SCF).

Instances A particular instantiation of the service. See
scf_instance_create(3SCF).

A service groups one or more related instances and provides a shared configuration for them.

An scf_service_t is an opaque handle that can be set to a single service at any given time.
The scf_service_create() function allocates and initializes a new scf_service_t bound to
handle. The scf_service_destroy() function destroys and frees svc.

The scf_service_handle() function retrieves the handle to which svc is bound.

The scf_service_get_parent() function sets sc to the scope that is the parent of svc.

The scf_service_get_name() function retrieves the name of the service to which svc is set.

The scf_scope_get_service() function sets svc to the service specified by name in the scope
specified by sc.
The scf_scope_add_service() function sets svc to a new service specified by name in the scope specified by sc.

The scf_service_delete() function deletes the service to which svc is set, as well as all of its children.

**Return Values**

Upon successful completion, scf_service_create() returns a new scf_service_t. Otherwise, it returns NULL.

Upon successful completion, scf_service_handle() returns the handle to which svc is bound. Otherwise, it returns NULL.

Upon successful completion, scf_service_get_name() returns the length of the string written, not including the terminating null byte. Otherwise, it returns -1.

Upon successful completion, scf_service_get_parent(), scf_scope_get_service(), scf_scope_add_service(), and scf_service_delete() return 0. Otherwise, it returns -1.

**Errors**

The scf_service_create() function will fail if:

- SCF_ERROR_INVALID_ARGUMENT  The value of the handle argument is NULL.
- SCF_ERROR_NO_MEMORY         There is not enough memory to allocate an scf_service_t.
- SCF_ERROR_NO_RESOURCES      The server does not have adequate resources for a new scope handle.

The scf_service_handle() function will fail if:

- SCF_ERROR_HANDLE_DESTROYED  The handle associated with svc has been destroyed.

The scf_service_get_name(), scf_service_get_parent(), and scf_service_delete() functions will fail if:

- SCF_ERROR_DELETED           The service has been deleted by someone else.
- SCF_ERROR_NOT_SET           The service is not set.
- SCF_ERROR_NOT_BOUND         The handle is not bound.
- SCF_ERROR_CONNECTION_BROKEN  The connection to the repository was lost.

The scf_service_delete() function will fail if:

- SCF_ERROR_EXISTS            The service contains instances.
- SCF_ERROR_NO_RESOURCES      The server does not have adequate resources for a new scope handle.

The scf_scope_add_service() function will fail if:

The scf_scope_get_service() function will fail if:

SCF_ERROR_BACKEND_ACCESS: The storage mechanism that the repository server (svc.configd(1M)) chose for the operation denied access.

SCF_ERROR_INTERNAL: An internal error occurred.

The scf_scope_add_service() and scf_scope_get_service() functions will fail if:

SCF_ERROR_CONNECTION_BROKEN: The connection to the repository was lost.

SCF_ERROR_DELETED: The parent entity has been deleted.

SCF_ERROR_HANDLE_MISMATCH: The scope and service are not derived from the same handle.

SCF_ERROR_INVALID_ARGUMENT: The value of the name argument is not a valid service name.

SCF_ERROR_NO_RESOURCES: The server does not have the resources to complete the request.

SCF_ERROR_NOT_BOUND: The handle is not bound.

SCF_ERROR_NOT_FOUND: The service specified by name was not found.

SCF_ERROR_NOT_SET: The scope is not set.

The scf_scope_add_service() and scf_service_delete() functions will fail if:

SCF_ERROR_PERMISSION_DENIED: The user does not have sufficient privileges to create or delete a service.

SCF_ERROR_BACKEND_READONLY: The repository backend is read-only.

SCF_ERROR_BACKEND_ACCESS: The repository backend refused the modification.

The scf_error(3SCF) function can be used to retrieve the error value.

Attributes: See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT- Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also: libscf(3LIB), scf_error(3SCF), scf_handle_decode_fmri(3SCF), scf_iter_service_pgs(3SCF), scf_iter_service_pgs_typed(3SCF),
scf_instance_create(3SCF), scf_pg_create(3SCF), scf_scope_create(3SCF),
scf_service_to_fmri(3SCF), attributes(5), smf(5)
Name  scf_service_delcust, scf_instance_delcust, scf_pg_delcust, scf_property_delcust – remove administrative customizations in the Service Configuration Facility

Synopsis  cc [ flag... ] file... -lsfc [ library... ]
#include <libscf.h>

int scf_service_delcust(scf_service_t *svc);
int scf_instance_delcust(scf_instance_t *inst);
int scf_pg_delcust(scf_propertygroup_t *pg);
int scf_transaction_property_delcust(scf_transaction_t *t,
    scf_transaction_entry_t *e, const char *prop);

Description  The *_delcust() family of commands allows libscf(3LIB) consumers to remove all administrative customizations for an entity with one function. This includes removing any “masked” notations previously created by using the scf_*_delete() functions.

The customizations at the administrative layer can be explored with the get_decoration suite of functions. See scf_service_get_decoration(3SCF).

The customizations removed are all customizations for the entity and its children, if any exist.

Return Values  Upon successful completion, these functions return 0. Otherwise they return —1.

Errors  These functions will fail if:

SCF_ERROR_DELETED
    The entity or one of its parents has been deleted by somebody else.

SCF_ERROR_NOT_SET
    The entity is not set.

SCF_ERROR_NOT_BOUND
    The repository handle is not bound.

SCF_ERROR_CONNECTION_BROKEN
    The connection to the repository was lost.

SCF_ERROR_NO_RESOURCES
    The server does not have adequate resources for a new scope handle.

SCF_ERROR_PERMISSION_DENIED
    The user does not have sufficient privileges delete customizations.

SCF_ERROR_BACKEND_READONLY
    The repository backend is read-only.

SCF_ERROR_BACKEND_ACCESS
    The repository backend refused the modification.

The scf_error(3SCF) function can be used to retrieve the error value.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libscf(3LIB), scf_error(3SCF), scf_handle_decode_fmri(3SCF), scf_service_get_decoration(3SCF), attributes(5)
Name  scf_service_is_masked, scf_instance_is_masked, scf_pg_is_masked, scf_property_is_masked  
   – determine whether entities are masked in the Service Configuration Facility

Synopsis  cc [ flag... ] file... -lscf [ library... ]
   #include <libscf.h>

   int scf_service_is_masked(const scf_service_t *svc);
   int scf_instance_is_masked(const scf_instance_t *inst);
   int scf_pg_is_masked(const scf_propertygroup_t *pg);
   int scf_property_is_masked(const scf_property_t *prop);

Description  Entities are masked if an administrator has requested they be deleted, but the entity is backed
   by configuration in the filesystem such as manifests and profiles.

   Masked entities are not discovered through libscf(3LIB) unless the caller decorates the
   initial handle with scf_handl_e_decorate(h, "show_masked", SCF_DECORATE_TRUE). If the
   caller needs to ignore masked entities, it must explicitly call one of the functions described on
   this manual page to determine the entity is masked.

   These functions allow consumers to determine whether an entity is masked.

Return Values  If the entity is not masked, these functions return 0. If the entity is masked, they return 1. In
   the event of error, they return -1.

Errors  These functions will fail if:

   SCF_ERROR_EXISTS                  The entity is not masked.
   SCF_ERROR_DELETED                 The entity or one of its parents has been deleted.
   SCF_ERROR_NOT_SET                 The entity is not set.
   SCF_ERROR_NOT_BOUND               The repository handle is not bound.
   SCF_ERROR_CONNECTION_BROKEN       The connection to the repository was lost.

   The scf_error(3SCF) function can be used to retrieve the error value.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>
See Also  libscf(3LIB), scf_error(3SCF), scf_handle_create(3SCF),
         scf_handle_decode_fmri(3SCF), scf_iter_create(3SCF), attributes(5)


**Name**

scf_simple_prop_get, scf_simple_prop_free, scf_simple_app_props_get,
scf_simple_app_props_free, scf_simple_app_props_next, scf_simple_app_props_search,
scf_simple_prop_numvalues, scf_simple_prop_type, scf_simple_prop_name,
scf_simple_prop_pgname, scf_simple_prop_next_boolean, scf_simple_prop_next_count,
scf_simple_prop_next_integer, scf_simple_prop_next_time, scf_simple_prop_next_astring,
scf_simple_prop_next_ustring, scf_simple_prop_next_opaque, scf_simple_prop_next_reset

Simplified property read interface to Service Configuration Facility

**Synopsis**

cc [ flag... ] file... -lsclf [ library... ]

```
#include <libscf.h>

scf_simple_prop_t *scf_simple_prop_get(scf_handle_t *handle,
    const char *instance, const char *pgname, const char *propname);

void scf_simple_prop_free(scf_simple_prop_t *prop);

scf_simple_app_props_t *scf_simple_app_props_get(scf_handle_t *handle,
    const char *instance);

void scf_simple_app_props_free(scf_simple_app_props_t *propblock);

const scf_simple_prop_t *scf_simple_app_props_next
    (const scf_simple_app_props_t *propblock, scf_simple_prop_t *last);

const scf_simple_prop_t *scf_simple_app_props_search
    (const scf_simple_app_props_t *propblock, const char *pgname,
    const char *propname);

ssize_t scf_simple_prop_numvalues(const scf_simple_prop_t *prop);

scf_type_t scf_simple_prop_type(const scf_simple_prop_t *prop);

const char *scf_simple_prop_name(const scf_simple_prop_t *prop);

const char *scf_simple_prop_pgname(const scf_simple_prop_t *prop);

uint8_t *scf_simple_prop_next_boolean(const scf_simple_prop_t *prop);

uint64_t *scf_simple_prop_next_count(const scf_simple_prop_t *prop);

int64_t *scf_simple_prop_next_integer(const scf_simple_prop_t *prop);

int64_t *scf_simple_prop_next_time(const scf_simple_prop_t *prop,
    int32_t *nsec);

char *scf_simple_prop_next_astring(const scf_simple_prop_t *prop);

char *scf_simple_prop_next_ustring(const scf_simple_prop_t *prop);

void *scf_simple_prop_next_opaque(const scf_simple_prop_t *prop,
    size_t *length);

void *scf_simple_prop_next_reset(const scf_simple_prop_t *prop);
```
The simplified read interface to the Service Configuration Facility deals with properties and blocks of properties.

The `scf_simple_prop_get()` function pulls a single property. The `scf_simple_prop_*()` functions operate on the resulting `scf_simple_prop_t`.

The application might need to get many properties or iterate through all properties. The `scf_simple_app_props_get()` function gets all properties from the service instance that are in property groups of type 'application'. Individual properties are pulled from the block using the `scf_simple_app_props_next()` function for iteration or `scf_simple_app_props_search()` to search. The pointer to the `scf_simple_prop_t` returned from iteration or searching can be acted upon using the `scf_simple_prop_*()` functions. Each `scf_*` get() function has an accompanying `scf_*_free` function. The application does not free the pointer to the `scf_simple_prop_t` returned from the `scf_simple_app_props_next()` and `scf_simple_app_props_search()` calls. A free call is only used with a corresponding get call.

The `scf_simple_prop_*()` functions return references to the read-only in-memory copy of the property information. Any changes to this information results in unstable behavior and inaccurate results. The simplified read interface provides read access only, with no provisions to modify data in the service configuration facility repository.

The `scf_simple_prop_get()` function takes as arguments a bound handle, a service instance FMRI, and the property group and property name of a property. If `handle` is `NULL`, the library uses a temporary handle created for the purpose. If `instance` is `NULL` the library automatically finds the FMRI of the calling process. If `pgname` is `NULL`, the library uses the default application property group. The caller is responsible for freeing the returned property with `scf_simple_prop_free()`.

The `scf_simple_prop_free()` function frees the `scf_simple_prop_t` allocated by `scf_simple_prop_get()`.

The `scf_simple_app_props_get()` function takes a bound handle and a service instance FMRI and pulls all the application properties into an `scf_simple_app_props_t`. If `handle` is `NULL`, the library uses a temporary handle created for the purpose. If `instance` is `NULL`, the library looks up the instance FMRI of the process calling the function. The caller is responsible for freeing the `scf_simple_app_props_t` with `scf_simple_app_props_free()`.

The `scf_simple_app_props_free()` function frees the `scf_simple_app_props_t` allocated by `scf_simple_app_props_get()`.

The `scf_simple_app_props_next()` function iterates over each property in an `scf_simple_app_props_t`. It takes an `scf_simple_app_props_t` pointer and the last property returned from the previous call and returns the next property in the `scf_simple_app_props_t`. Because the property is a reference into the `scf_simple_app_props_t`, its lifetime extends only until that structure is freed.
The `scf_simple_app_props_search()` function queries for an exact match on a property in a
property group. It takes an apps prop object, a property group name, and a property name,
and returns a property pointer. Because the property is a reference into the
`scf_simple_app_props_t`, its lifetime extends only until that structure is freed. If the
property group name, `pgname`, is NULL, “application” is used.

The `scf_simple_prop_numvalues()` function takes a pointer to a property and returns the
number of values in that property.

The `scf_simple_prop_type()` function takes a pointer to a property and returns the type of
the property in an `scf_type_t`.

The `scf_simple_prop_name()` function takes a pointer to a property and returns a pointer to
the property name string.

The `scf_simple_prop_pgname()` function takes a pointer to a property and returns a pointer
to the property group name string. The `scf_simple_prop_next_boolean()`,
`scf_simple_prop_next_count()`, `scf_simple_prop_next_integer()`,
`scf_simple_prop_next_astring()`, and `scf_simple_prop_next_ustring()` functions take
a pointer to a property and return the first value in the property. Subsequent calls iterate over
all the values in the property. The property's internal iteration can be reset with
`scf_simple_prop_next_reset()`.

The `scf_simple_prop_next_time()` function takes a pointer to a property and the address of
an allocated `int32_t` to hold the nanoseconds field, and returns the first value in the property.
Subsequent calls iterate over the property values.

The `scf_simple_prop_next_opaque()` function takes a pointer to a property and the address
of an allocated integer to hold the size of the opaque buffer. It returns the first value in the
property. Subsequent calls iterate over the property values, as do the
`scf_simple_prop_next_*()` functions. The `scf_simple_prop_next_opaque()` function
writes the size of the opaque buffer into the allocated integer.

The `scf_simple_prop_next_reset()` function resets iteration on a property, so that a call to
one of the `scf_simple_prop_next_*()` functions returns the first value in the property.

**Return Values**

Upon successful completion, `scf_simple_prop_get()` returns a pointer to an allocated
`scf_simple_prop_t`. Otherwise, it returns `NULL`.

Upon successful completion, `scf_simple_app_props_get()` returns a pointer to an allocated
`scf_simple_app_props_t`. Otherwise, it returns `NULL`.

Upon successful completion, `scf_simple_app_props_next()` returns a pointer to an
`scf_simple_prop_t`. Otherwise, it returns `NULL`.

Upon successful completion, `scf_simple_app_props_search()` returns a pointer to an
`scf_simple_prop_t`. Otherwise, it returns `NULL`.

Return Values
Upon successful completion, scf_simple_prop_numvalues() returns the number of values in a property. Otherwise, it returns -1.

Upon successful completion, scf_simple_prop_type() returns an scf_type_t. Otherwise, it returns -1.

Upon successful completion, scf_simple_prop_name() and scf_simple_prop_pgname() return character pointers. Otherwise, they return NULL.

Upon successful completion, scf_simple_prop_next_boolean(), scf_simple_prop_next_count(), scf_simple_prop_next_integer(), scf_simple_prop_next_time(), scf_simple_prop_next_astring(), scf_simple_prop_next_ustring(), and scf_simple_prop_next_opaque() return a pointer to the next value in the property. After all values have been returned, NULL is returned and SCF_ERROR_NONE is set. On failure, NULL is returned and the appropriate error value is set.

Errors

The scf_simple_prop_get() and scf_simple_app_props_get() functions will fail if:

SCF_ERROR_CONNECTION_BROKEN  The connection to the datastore is broken.
SCF_ERROR_INVALID_ARGUMENT   The instance FMRI is invalid or property name is NULL.
SCF_ERROR_NO_MEMORY          The memory allocation failed.
SCF_ERROR_NOT_BOUND          The connection handle is not bound.
SCF_ERROR_NOT_FOUND          The specified instance or property does not exist.
SCF_ERROR_PERMISSION_DENIED The caller is not authorized to read the property's value(s).

The scf_simple_app_props_next() function will fail if:

SCF_ERROR_NOT_SET   The value of the propblock argument is NULL.

The scf_simple_app_props_search() function will fail if:

SCF_ERROR_NOT_FOUND The property was not found.
SCF_ERROR_NOT_SET   The value of the propblock or propname argument is NULL.

The scf_simple_prop_numvalues(), scf_simple_prop_type(), scf_simple_prop_name(), and scf_simple_prop_pgname() functions will fail if:

SCF_ERROR_NOT_SET   The property is NULL.

The scf_simple_prop_next_boolean(), scf_simple_prop_next_count(), scf_simple_prop_next_integer(), scf_simple_prop_next_time(), scf_simple_prop_next_astring(), scf_simple_prop_next_ustring(), and scf_simple_prop_next_opaque() functions will fail if:

SCF_ERROR_NOT_SET   The property is NULL.
SCF_ERROR_TYPE_MISMATCH  The requested type does not match the property type.

**Examples**

**EXAMPLE 1**  Simple Property Get

```c
/*
 * In this example, we pull the property named "size" from the
 * default property group. We make sure that the property
 * isn't empty, and then copy it into the sizeval variable.
 */

scf_simple_prop_t *prop;
ssize_t numvals;
int64_t *sizeval;

prop = scf_simple_prop_get(
    "svc://localhost/category/service/instance",
    NULL, "size");

numvals = scf_simple_prop_numvalues(prop);

if(numvals > 0){
    sizeval = scf_simple_prop_next_integer(prop);
}

scf_simple_prop_free(prop);
```

**EXAMPLE 2**  Property Iteration

```c
scf_simple_prop_t *prop;
scf_simple_app_props_t *appprops;

appprops = scf_simple_app_props_get(
    "svc://localhost/category/service/instance");

prop = scf_simple_app_props_next(appprops, NULL);

while(prop != NULL)
{
    /*
     * This iteration will go through every property in the
     * instance’s application block. The user can use
     * the set of property functions to pull the values out
     * of prop, as seen in other examples.
     */

    (...code acting on each property...)
```
EXAMPLE 2  Property Iteration  (Continued)

    prop = scf_simple_app_props_next(appprops, prop);
    
    }

scf_simple_app_props_free(appprops);

EXAMPLE 3  Property Searching

    /*
    * In this example, we pull the property block from the instance,
    * and then query it. Generally speaking, the simple get would
    * be used for an example like this, but for the purposes of
    * illustration, the non-simple approach is used. The property
    * is a list of integers that are pulled into an array.
    * Note how val is passed back into each call, as described above.
    */

    approps = scf_simple_app_props_get(  
        "svc://localhost/category/service/instance";
    
    prop = scf_simple_app_props_search(appprops, "appname", "numlist");

    if(prop != NULL){
        
        numvals = scf_simple_prop_numvalues(prop);
        
        if(numvals > 0){
            
            intlist = malloc(numvals * sizeof(int64_t));
            
            val = scf_simple_prop_next_integer(prop);
            
            for(i=0, i < numvals, i++){
                intlist[i] = *val;
                val = scf_simple_prop_next_integer(prop);
            }
            
        }
    }

    scf_simple_app_props_free(appprops);

Extended Library Functions, Volume 4  169
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libscf(3LIB), scf_error(3SCF), attributes(5)
# scf_simple_walk_instances

## Name
scf_simple_walk_instances – observational interface for Service Configuration Facility

## Synopsis
```c
cc [ flag... ] file... -lscf [ library... ]
#include <libscf.h>

int scf_simple_walk_instances(uint_t flags, void *private,
    int (*inst_callback)(scf_handle_t *, scf_instance_t *, void *));
```

## Description
The `scf_simple_walk_instances()` function iterates over every service instance in a specified state and calls a callback function provided by the user on each specified instance.

The function takes a `flags` argument to indicate which instance states are involved in the iteration, an opaque buffer to be passed to the callback function, and a callback function with three arguments, a handle, an instance pointer, and an opaque buffer. If the callback function returns a value other than success, iteration is ended, an error is set, and the function returns -1.

The handle passed to the callback function is provided to the callback function by the library. This handle is used by the callback function for all low-level allocation involved in the function.

The simplified library provides defined constants for the `flags` argument. The user can use a bitwise OR to apply more than one flag. The `SCF_STATE_ALL` flag is a bitwise OR of all the other states. The flags are:

- `SCF_STATE_UNINIT`
- `SCF_STATE_MAINT`
- `SCF_STATE_OFFLINE`
- `SCF_STATE_DISABLED`
- `SCF_STATE_ONLINE`
- `SCF_STATE_DEGRADED`
- `SCF_STATE_ALL`

## Return Values
Upon successful completion, `scf_simple_walk_instances()` returns 0. Otherwise, it returns -1.

## Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
`libscf(3LIB), attributes(5)}`
scf_snaplevel_create(3SCF)

**Name**
scf_snaplevel_create, scf_snaplevel_handle, scf_snaplevel_destroy, scf_snaplevel_get_parent, scf_snaplevel_get_scope_name, scf_snaplevel_get_service_name, scf_snaplevel_get_instance_name, scf_snapshot_get_base_snaplevel, scf_snaplevel_get_next_snaplevel – create and manipulate snaplevel handles in the Service Configuration Facility

**Synopsis**
cc [ flag... ] file... -lscf [ library... ]
#include <libscf.h>

scf_snaplevel_t *scf_snaplevel_create(scf_handle_t *handle);
scf_snaplevel_t *scf_snaplevel_handle(scf_snaplevel_t *level);
void scf_snaplevel_destroy(scf_snaplevel_t *level);
int scf_snaplevel_get_parent(const scf_snaplevel_t *level,
                           const scf_snapshot_t *snap);
ssize_t scf_snaplevel_get_scope_name(const scf_snaplevel_t *level,
                          char *buf, size_t size);
ssize_t scf_snaplevel_get_service_name(const scf_snaplevel_t *level,
                          char *buf, size_t size);
ssize_t scf_snaplevel_get_instance_name(const scf_snaplevel_t *level,
                          char *buf, size_t size);
int scf_snapshot_get_base_snaplevel(const scf_snapshot_t *snap,
                         scf_snaplevel_t *level);
int scf_snaplevel_get_next_snaplevel(scf_snaplevel_t *in,
                          scf_snaplevel_t *out);

**Description**
A snaplevel holds all of the property groups associated with either a service or an instance. Each snapshot has an ordered list of snaplevels. Snaplevels contain the names of the instance or service from which they are derived.

An scf_snaplevel_t is an opaque handle that can be set to a single snaplevel at any given time. When set, the scf_snaplevel_t inherits the point in time from the scf_snapshot_t from which it comes.

The scf_snaplevel_create() function allocates and initializes a new scf_snaplevel_t bound to handle. The scf_snaplevel_destroy() function destroys and frees level.

The scf_snaplevel_handle() function retrieves the handle to which level is bound.

The scf_snaplevel_get_parent() function sets snap to the parent snapshot of the snaplevel to which level is set. The snapshot specified by snap is attached to the same point in time as level.

The scf_snaplevel_get_scope_name(), scf_snaplevel_get_service_name(), and scf_snaplevel_get_instance_name() functions retrieve the name of the scope, service, and
instance for the snapshot to which snap is set. If the snaplevel is from an instance, all three succeed. If the snaplevel is from a service, scf_snaplevel_get_instance_name() fails.

The scf_snapshot_get_base_snaplevel() function sets level to the first snaplevel in the snapshot to which snap is set. The scf_snaplevel_get_next_snaplevel() function sets out to the next snaplevel after the snaplevel to which in is set. Both the in and out arguments can point to the same scf_snaplevel_t.

To retrieve the property groups associated with a snaplevel, see scf_iter_snaplevel_pgs(3SCF), scf_iter_snaplevel_pgs_typed(3SCF), and scf_snaplevel_get_pg(3SCF).

**Return Values**

Upon successful completion, scf_snaplevel_create() returns a new scf_snaplevel_t. Otherwise, it returns NULL.

Upon successful completion, scf_snaplevel_get_scope_name(), scf_snaplevel_get_service_name(), and scf_snaplevel_get_instance_name() return the length of the string written, not including the terminating null byte. Otherwise, they return -1.

Upon successful completion, scf_snaplevel_get_parent(), scf_snapshot_get_base_snaplevel(), and scf_snaplevel_get_next_snaplevel() return. Otherwise, they return -1.

**Errors**

The scf_snaplevel_create() function will fail if:

- SCF_ERROR_INVALID_ARGUMENT  The handle argument is NULL.
- SCF_ERROR_NO_MEMORY  There is not enough memory to allocate an scf_snaplevel_t.
- SCF_ERROR_NO_RESOURCES  The server does not have adequate resources for a new snapshot handle.

The scf_snaplevel_get_scope_name(), scf_snaplevel_get_service_name(), scf_snaplevel_get_instance_name(), and scf_snaplevel_get_parent() functions will fail if:

- SCF_ERROR_DELETED  The object referred to by level has been deleted.
- SCF_ERROR_NOT_SET  The snaplevel is not set.
- SCF_ERROR_NOT_BOUND  The handle is not bound.
- SCF_ERROR_CONNECTION_BROKEN  The connection to the repository was lost.

The scf_snaplevel_get_instance_name() function will fail if:

- SCF_ERROR_CONSTRAINT_VIOLATED  The snaplevel is derived from a service.

The scf_snapshot_get_base_snaplevel() function will fail if:
The connection to the repository was lost.
The snapshot has been deleted.
The snapshot and snaplevel are not derived from the same handle.
The server does not have the resources to complete the request.
The handle is not bound.
There are no snaplevels in this snapshot.
The snapshot is not set.

The scf_snaplevel_get_next_snaplevel() function will fail if:
The snaplevel has been deleted.
The snaplevel is not set.
The in and out arguments are not derived from the same handle.
The handle is not bound.
The connection to the repository was lost.
There are no more snaplevels in this snapshot.

The scf_error(3SCF) function can be used to retrieve the error value.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTEVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also libcfs(3LIB), scf_error(3SCF), scf_iter_snaplevel_pgs(3SCF),
scf_iter_snaplevel_pgs_typed(3SCF), scf_snaplevel_get_pg(3SCF), attributes(5)
**Name**  
scf_snapshot_create, scf_snapshot_handle, scf_snapshot_destroy, scf_snapshot_get_parent,  
scf_snapshot_get_name, scf_snapshot_update, scf_instance_get_snapshot – create and  
manipulate snapshot handles and snapshots in the Service Configuration Facility

**Synopsis**  
cc [ flag... ] file... -lsxcf [ library... ]  
#include <libscf.h>  
scf_snapshot_t *scf_snapshot_create(scf_handle_t *handle);  
scf_handle_t *scf_snapshot_handle(scf_snapshot_t *snap);  
void scf_snapshot_destroy(scf_snapshot_t *snap);  
int scf_snapshot_get_parent(const scf_snapshot_t *snap,  
   scf_instance_t *inst);  
ssize_t scf_snapshot_get_name(const scf_snapshot_t *snap,  
   char *buf, size_t size);  
int scf_snapshot_update(scf_snapshot_t *snap);  
int scf_instance_get_snapshot(const scf_instance_t *inst,  
   const char *name, scf_snapshot_t *snap);

**Description**  
A snapshot is an unchanging picture of the full set of property groups associated with an  
instance. Snapshots are automatically created and managed by the Solaris Management  
Facility. See smf(5).

A snapshot consists of a set of snaplevels, each of which holds copies of the property groups  
associated with an instance or service in the resolution path of the base instance. Typically,  
there is one snaplevel for the instance and one for the instance’s parent service.

The `scf_snapshot_create()` function allocates and initializes a new `scf_snapshot_t` bound  
to `handle`. The `scf_snapshot_destroy()` function destroys and frees `snap`.

The `scf_snapshot_handle()` function retrieves the handle to which `snap` is bound.

The `scf_snapshot_get_parent()` function sets `inst` to the parent of the snapshot to which  
`snap` is set.

The `scf_snapshot_get_name()` function retrieves the name of the snapshot to which `snap` is  
set.

The `scf_snapshot_update()` function reattaches `snap` to the latest version of the snapshot to  
which `snap` is set.

The `scf_instance_get_snapshot()` function sets `snap` to the snapshot specified by `name` in  
the instance specified by `inst`. To walk all of the snapshots, see `scf_iter_instance_snapshots(3SCF)`.

To access the snaplevels of a snapshot, see `scf_snapshot_get_base_snaplevel(3SCF)`.
Return Values  Upon successful completion, scf_snapshot_create() returns a new scf_snapshot_t. Otherwise, it returns NULL.

Upon successful completion, scf_snapshot_handle() returns the handle to which snap is bound. Otherwise, it returns NULL.

Upon successful completion, scf_snapshot_get_name() returns the length of the string written, not including the terminating null byte. Otherwise, it returns NULL.

The scf_snapshot_update() function returns 1 if the snapshot was updated, 0 if the snapshot had not been updated, and -1 on failure.

Upon successful completion, scf_snapshot_get_parent() and scf_instance_get_snapshot() return 0. Otherwise, they return -1.

Errors  The scf_snapshot_create() function will fail if:

SCF_ERROR_INVALID_ARGUMENT
  The handle argument is NULL.

SCF_ERROR_NO_MEMORY
  There is not enough memory to allocate an scf_snapshot_t.

SCF_ERROR_NO_RESOURCES
  The server does not have adequate resources for a new instance handle.

The scf_snapshot_handle() function will fail if:

SCF_ERROR_HANDLE_DELETED
  The handle associated with snap has been destroyed.

The scf_snapshot_get_name() and scf_snapshot_get_parent() functions will fail if:

SCF_ERROR_DELETED
  The snapshot has been deleted.

SCF_ERROR_NOT_SET
  The snapshot is not set.

SCF_ERROR_NOT_BOUND
  The handle is not bound.

SCF_ERROR_CONNECTION_BROKEN
  The connection to the repository was lost.

The scf_snapshot_update() function will fail if:

SCF_ERROR_CONNECTION_BROKEN
  The connection to the repository was lost.

SCF_ERROR_DELETED
  An ancestor of the snapshot specified by snap has been deleted.
SCF_ERROR_INTERNAL
   An internal error occurred. This can happen if snap has been corrupted.

SCF_ERROR_INVALID_ARGUMENT
   The snap argument refers to an invalid scf_snapshot_t.

SCF_ERROR_NOT_BOUND
   The handle is not bound.

SCF_ERROR_NOT_SET
   The snapshot specified by snap is not set.

The scf_instance_get_snapshot() function will fail if:

SCF_ERROR_BACKEND_ACCESS
   The storage mechanism that the repository server (svc.configd(1M)) chose for the
   operation denied access.

SCF_ERROR_CONNECTION_BROKEN
   The connection to the repository was lost.

SCF_ERROR_DELETED
   The instance has been deleted.

SCF_ERROR_HANDLE_MISMATCH
   The instance and snapshot are not derived from the same handle.

SCF_ERROR_INTERNAL
   An internal error occurred.

SCF_ERROR_INVALID_ARGUMENT
   The value of the name argument is not a valid snapshot name.

SCF_ERROR_NO_RESOURCES
   The server does not have the resources to complete the request.

SCF_ERROR_NOT_BOUND
   The handle is not bound.

SCF_ERROR_NOT_FOUND
   The snapshot specified by name was not found.

SCF_ERROR_NOT_SET
   The instance is not set.

The scf_error(3SCF) function can be used to retrieve the error value.

Attributes
   See attributes(5) for descriptions of the following attributes:
### scf_snapshot_create(3SCF)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also: libscf(3LIB), scf_error(3SCF), scf_iter_instance_snapshots(3SCF), scf_snapshot_get_base_snaplevel(3SCF), attributes(5), smf(5)
scf_tmpl_pg_create(3SCF)

**Name**

scf_tmpl_pg_create, scf_tmpl_pg_reset, scf_tmpl_pg_destroy, scf_tmpl_get_by_pg_name, scf_tmpl_get_by_pg, scf_tmpl_iter_pgs – template property group functions

**Synopsis**

cc [ flag... ] file... -lscf [ library... ]

#include <libscf.h>

scf_pg_tmpl_t *scf_tmpl_pg_create(scf_handle_t *handle);
void scf_tmpl_pg_reset(scf_pg_tmpl_t *pg_tmpl);
void scf_tmpl_pg_destroy(scf_pg_tmpl_t *pg_tmpl);
int scf_tmpl_get_by_pg_name(const char *instance_fmri,
                             const char *snapshot, const char *pg_name,
                             const char *pg_type, scf_pg_tmpl_t *pg_tmpl, int flags);
int scf_tmpl_get_by_pg(scf_propertygroup_t *pg,
                       scf_pg_tmpl_t *pg_tmpl, int flags)
int scf_tmpl_iter_pgs(scf_pg_tmpl_t *pg_tmpl, const char *fmri,
                       const char *snapshot, const char *pg_type, int flags);

**Description**

The template property group functions locate and give access to metadata about SMF configuration for service instances. They are used to directly access property group metadata and explore metadata for properties contained in those property groups.

A property group does not need to be currently defined in order to explore metadata about it, as long as the metadata is defined. Thus, the property group template functions operate on strings rather than `scf_propertygroup_t` entities.

By default, when an instance FMRI is specified, `scf_tmpl_get_by_pg_name()` and `scf_tmpl_iter_pgs()` lookup composed data from the running snapshot of the instance. A different snapshot may be explicitly selected by specifying a valid snapshot name rather than NULL for the snapshot argument. If a service FMRI is specified, the current properties are used.

By default, these functions also explore template data defined by the service or instance itself, the service’s restarter, and global template data. See `smf_template(5)` for more information about this composition.

Once retrieved, the `scf_pg_tmpl_t` can be explored using the `scf_tmpl_pg_name(3SCF)` and `scf_tmpl_prop_create(3SCF)` suite of functions.

Before calling `scf_tmpl_get_by_pg()`, `scf_tmpl_get_by_pg_name()`, or `scf_tmpl_iter_pgs()`, the `scf_pg_tmpl_t` must be allocated by `scf_tmpl_pg_create()`. The `scf_pg_tmpl_t` can be reset to contain no template information with `scf_tmpl_pg_reset()`, so that it can be used to start an iteration from scratch. All associated memory can be freed with `scf_tmpl_pg_destroy()`.
The `scf_tmpl_get_by_pg()` function locates the most specific matching template for the property group supplied. The parent of that property group can be either a service or an instance.

The `scf_tmpl_get_by_pg_name()` function locates the most specific matching template for the property group as specified. As described above, when the snapshot argument is NULL the default running snapshot is used. If flags includes `SCF_PG_TMPL_FLAG_CURRENT`, the snapshot argument is ignored and the current configuration is used. If flags includes `SCF_PG_TMPL_FLAG_EXACT`, only the exact FMRI is looked up. Either or both of the `pg_name` and `pg_type` arguments may be specified as NULL. In this case, `pg_name` and/or `pg_type` is wildcarded and matches any value. The most specific snapshot matching those arguments is returned.

The `scf_tmpl_iter_pgs()` function iterates across all templates defined for the specified FMRI, snapshot, and optional property group type. It also takes an optional flags argument. If flags includes `SCF_PG_TMPL_FLAG_CURRENT`, the snapshot argument is ignored and the "running" snapshot is used. `SCF_PG_TMPL_FLAG_REQUIRED` searches only for required property groups. `SCF_PG_TMPL_FLAG_EXACT` looks only at the exact FMRI provided for templates, and not for templates defined on its restarter or globally.

The iterator state for `scf_tmpl_iter_pgs()` is stored on the template data structure. The data structure should be allocated with `scf_tmpl_pg_create()` and to continue the iteration the previously returned structure should be passed in as an argument.

**Return Values**

The `scf_tmpl_pg_create()` function returns NULL on failure and a pointer to an allocated and populated `scf_pg_tmpl_t` on success. The caller is responsible for freeing the memory with `scf_tmpl_pg_destroy()`.

The `scf_tmpl_get_by_pg()` and `scf_tmpl_get_by_pg_name()` functions return 0 on success and -1 on failure.

The `scf_tmpl_iter_pgs()` function returns 1 for successful iteration to the next item, 0 for iteration successfully reaching its end, and —1 on error.

**Errors**

The `scf_tmpl_get_by_pg()`, `scf_tmpl_get_by_pg_name()`, and `scf_tmpl_iter_pgs()` functions will fail if:

- `SCF_ERROR_BACKEND_ACCESS` The storage mechanism that the repository server (``svc.configd``) chose for the operation denied access.

- `SCF_ERROR_CONNECTION_BROKEN` The connection to the repository was lost.

- `SCF_ERROR_DELETED` The instance or its template property group has been deleted.

- `SCF_ERROR_HANDLE_DESTROYED` The handle passed in has been destroyed.

- `SCF_ERROR_INTERNAL` An internal error occurred.
The `scf_tmpl_get_by_pg()` function will fail if:

**SCF_ERROR_NOT_SET** The property group specified by `pg` is not set.

The `scf_tmpl_pg_create()` function will fail if:

**SCF_ERROR_INVALID_ARGUMENT** The handle argument is NULL.

**SCF_ERROR_NO_MEMORY** There is no memory available.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

### See Also

`sfc.configd(1M), scf_tmpl_pg_name(3SCF), scf_tmpl_prop_create(3SCF), attributes(5), smf_template(5)`
The functions retrieve the metadata about a specific property group. They require that the template for the property group has already been located by one of the `scf_tmpl_pg_create(3SCF)` suite of functions.

The `scf_tmpl_pg_name()` function retrieves the name of the property group template and place it in `*out`. If the property group name is implicitly wildcarded (see `smf_template(5)`) in the template, this function will return a string containing `SCF_TMPL_WILDCARD` (“*”) in `*out`. The caller is responsible for freeing the `*out` buffer on success.

The `scf_tmpl_pg_type()` function will retrieve the type of the property group template and place it in `*out`. If the property group type is implicitly wildcarded (see `smf_template(5)`) in the template, this function will return a string containing `SCF_TMPL_WILDCARD` (“*”) in `*out`. The caller is responsible for freeing the `*out` buffer on success.

The `scf_tmpl_pg_target()` function will retrieve the target of the property group template and place it in `*out`. The caller is responsible for freeing the `*out` buffer on success.

The `scf_tmpl_pg_required()` function will determine whether the property group is required and place the result of that check in `*out`. If required is unset, `out` will be the default value of 0. If the property is explicitly set to required, `out` will be 1.

The `scf_tmpl_pg_common_name()` function will retrieve the property group’s localized common name as currently templated and place it in `*out`. A locale (as described in `setlocale(3C)`) may be specified, or if the supplied locale is NULL, the current locale will be used. If a common_name in the specified locale is not found, the function will also look for a
common_name in the C locale. Some templates will not specify the property group common name. The caller is responsible for freeing the *out buffer on success.

The scf_tmpl_pg_description() function will retrieve the property group's localized description as currently templated and place it in *out. A locale (as described in setlocale(3C)) may be specified, or if the supplied locale is NULL, the current locale will be used. If a description in the specified locale is not found, the function will also look for a description in the C locale. Some templates will not specify the property group description. The caller is responsible for freeing the *out buffer on success.

Return Values
Upon successful completion, scf_tmpl_pg_name(), scf_tmpl_pg_common_name(), scf_tmpl_pg_description(), scf_tmpl_pg_target(), and scf_tmpl_pg_type() return the length of the string written, not including the terminating null byte. Otherwise, they return -1.

Errors
The scf_tmpl_pg_name(), scf_tmpl_pg_common_name(), scf_tmpl_pg_description(), scf_tmpl_pg_required(), scf_tmpl_pg_target(), and scf_tmpl_pg_type() functions will fail if:

SCF_ERROR_BACKEND_ACCESS The storage mechanism that the repository server (svc.configd(1M)) chose for the operation denied access.

SCF_ERROR_CONNECTION_BROKEN The connection to the repository was lost.

SCF_ERROR_DELETED The template property group has been deleted.

SCF_ERROR_HANDLEDestroyed The handle passed in has been destroyed.

SCF_ERROR_INTERNAL An internal error occurred.

SCF_ERROR_NO_MEMORY There is not enough memory to populate the scf_pg_tmpl_t.

SCF_ERROR_NO_RESOURCES The server does not have adequate resources to complete the request.

SCF_ERROR_NOT_BOUND The handle is not currently bound.

SCF_ERROR_PERMISSION_DENIED The template could not be read due to access restrictions.

SCF_ERROR_TEMPLATE_INVALID The template data is invalid.

The scf_tmpl_pg_common_name() and scf_tmpl_pg_description() functions will fail if:

SCF_ERROR_NOT_FOUND The property does not exist or exists and has no value.

SCF_ERROR_INVALID_ARGUMENT The locale string is too long.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  svc.configd(1M), scf_tmpl_pg_create(3SCF), setlocale(3C), attributes(5), smf_template(5)
Template property functions locate and give access to metadata about properties. They require that the template for the property group containing the property has already been located by one of the `scf_tmpl_pg_create` (3SCF) suite of functions.

Once retrieved, the `scf_prop_tmpl_t` can be explored using the `scf_tmpl_prop_name` (3SCF) suite of functions.

Before calling `scf_tmpl_get_by_prop()` or `scf_tmpl_iter_props()`, the `scf_prop_tmpl_t` must be allocated by `scf_tmpl_prop_create`. The `scf_prop_tmpl_t` can be reset to contain no template information with `scf_tmpl_prop_reset()`, so that it can be used to start an iteration from scratch. All associated memory can be freed with `scf_tmpl_prop_destroy()`.

The `scf_tmpl_get_by_prop()` function locates template data about the property name specified.

The `scf_tmpl_iter_props()` function iterates across all property templates defined in the specified property group template.

The iterator state for `scf_tmpl_iter_props()` is stored on the property template data structure. The data structure should be allocated with `scf_tmpl_prop_create()`, and to continue the iteration the previously returned structure should be passed in as an argument.

The `scf_tmpl_get_by_prop()` function returns -1 on failure and 0 on success.

The `scf_tmpl_iter_props()` function returns 0 for successful iteration to the next item, 1 for iteration successfully reaching its end, and —1 on error.

The `scf_tmpl_get_by_prop()` and `scf_tmpl_iter_props()` functions will fail if:

**SCF_ERROR_BACKEND_ACCESS**  
The storage mechanism that the repository server (svc.configd(1M)) chose for the operation denied access.

**SCF_ERROR_CONNECTION_BROKEN**  
The connection to the repository was lost.
The instance or its template property group has been deleted.

SCF_ERROR_HANDLE_DESTROYED  The handle passed in has been destroyed.

SCF_ERROR_INTERNAL  An internal error occurred.

SCF_ERROR_INVALID_ARGUMENT  One of the arguments is invalid.

SCF_ERROR_NO_MEMORY  There is not enough memory to populate the scf_prop_tmpl_t.

SCF_ERROR_NO_RESOURCES  The server does not have adequate resources to complete the request.

SCF_ERROR_NOT_BOUND  The handle is not currently bound.

The scf_tmpl_get_by_prop() function will fail if:

SCF_ERROR_NOT_FOUND  Template object matching property doesn’t exist in the repository.

SCF_ERROR_TYPE_MISMATCH  Matching template object is the wrong type in the repository.

SCF_ERROR_PERMISSION_DENIED  The template could not be read due to access restrictions.

SCF_ERROR_TEMPLATE_INVALID  The template data is invalid.

The scf_tmpl_prop_create() function will fail if:

SCF_ERROR_INVALID_ARGUMENT  The handle argument is NULL.

SCF_ERROR_NO_MEMORY  There is no memory available.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  svc.configd(1M), scf_tmpl_pg_create(3SCF), scf_tmpl_prop_name(3SCF), attributes(5), smf_template(5)
Name  scf_tmpl_prop_name, scf_tmpl_prop_type, scf_tmpl_prop_required,
scf_tmpl_prop_common_name, scf_tmpl_prop_description, scf_tmpl_prop_units,
scf_tmpl_prop_visibility, scf_tmpl_visibility_to_string, scf_tmpl_prop_cardinality,
scf_tmpl_prop_internal_seps, scf_tmpl_value_name_constraints, scf_count_ranges_destroy,
scf_int_ranges_destroy, scf_tmpl_value_count_range_constraints,
scf_tmpl_value_int_range_constraints, scf_tmpl_value_name_choices, scf_values_destroy,
scf_tmpl_value_count_range_choices, scf_tmpl_value_int_range_choices,
scf_tmpl_value_common_name, scf_tmpl_value_description, scf_tmpl_value_in_constraint
– retrieve the metadata about a specific property

Synopsis  cc [ flag... ] file... -lscf [ library... ]
#include <libscf.h>

ssize_t scf_tmpl_prop_name(const scf_prop_tmpl_t *prop_tmpl,
                          char **out);
int scf_tmpl_prop_type(const scf_prop_tmpl_t *prop_tmpl,
                       scf_type_t *out);
int scf_tmpl_prop_required(const scf_prop_tmpl_t *prop_tmpl,
                            uint8_t *out)
ssize_t scf_tmpl_prop_common_name(const scf_prop_tmpl_t *prop_tmpl,
                                   char *locale, char **out);
ssize_t scf_tmpl_prop_description(const scf_prop_tmpl_t *prop_tmpl,
                                   char *locale, char **out);
ssize_t scf_tmpl_prop_units(const scf_prop_tmpl_t *prop_tmpl,
                            const char *locale, char **out);
int scf_tmpl_prop_visibility(const scf_prop_tmpl_t *prop_tmpl,
                             uint8_t *out);
const char *scf_tmpl_visibility_to_string(uint8_t visibility);
int scf_tmpl_prop_cardinality(const scf_prop_tmpl_t *prop_tmpl,
                             uint64_t *min, uint64_t *max);
int scf_tmpl_prop_internal_seps(const scf_prop_tmpl_t *prop_tmpl,
                                 scf_values_t *out);
int scf_tmpl_value_name_constraints(const scf_prop_tmpl_t *prop_tmpl,
                                     scf_values_t *out);
void scf_count_ranges_destroy(scf_count_ranges_t *ranges);
void scf_int_ranges_destroy(scf_int_ranges_t *ranges);
int scf_tmpl_value_count_range_constraints(
                                          const scf_prop_tmpl_t *prop_tmpl,
                                          scf_count_ranges_t *ranges);
int scf_tmpl_value_int_range_constraints(
                                          const scf_prop_tmpl_t *prop_tmpl,
                                          scf_int_ranges_t *ranges);
int scf_tmpl_value_name_choices(const scf_prop_tmpl_t *prop_tmpl,
    scf_values_t *vals);

void scf_values_destroy(scf_values_t *vals);

int scf_tmpl_value_count_range_choices(
    const scf_prop_tmpl_t *prop_tmpl,
    scf_count_ranges_t *ranges);

int scf_tmpl_value_int_range_choices(const scf_prop_tmpl_t *prop_tmpl,
    scf_int_ranges_t *ranges);

ssize_t scf_tmpl_value_common_name(const scf_prop_tmpl_t *prop_tmpl,
    const char *locale, const char *value, char **out);

ssize_t scf_tmpl_value_description(const scf_prop_tmpl_t *prop_tmpl,
    const char *locale, const char *value, char **out);

int scf_tmpl_value_in_constraint(const scf_prop_tmpl_t *prop_tmpl,
    scf_value_t *value, scf_tmpl_errors_t **errs);

---

**Description**

These functions retrieve the metadata about a specific property. They require that the template for the property has already been located by one of the scf_tmpl_prop_create(3SCF) suite of functions.

The `scf_tmpl_prop_name()` function will retrieve the property's name as currently templated and place it in `*out`. The caller is responsible for freeing the `*out` buffer on success.

The `scf_tmpl_prop_type()` function will retrieve the type of the property as templated and place the type in `out`.

The `scf_tmpl_prop_required()` function will determine whether the property is required in this property group and place the result of that check in `out`. If required is unset, `out` will be the default, 0. If the property is explicitly set to required, `out` will be 1.

The `scf_tmpl_prop_common_name()` function will retrieve the property's localized common name as currently templated and place it in `*out`. A locale (as described in `setlocale(3C)`) can be specified, or if the supplied locale is NULL, the current locale will be used. If a common name in the specified locale is not found, the function will also look for a common name in the C locale. Some templates will not specify the property common name. The caller is responsible for freeing the `*out` buffer on success.

The `scf_tmpl_prop_description()` function will retrieve the property's localized description as currently templated and place it in `*out`. A locale (as described in `setlocale(3C)`) can be specified, or if the supplied locale is NULL, the current locale will be used. If a description in the specified locale is not found, the function will also look for a description in the C locale. Some templates will not specify the property description. The caller is responsible for freeing the `*out` buffer on success.

The `scf_tmpl_prop_visibility()` function will retrieve the visibility of the property as currently templated and place it in `out`. A property can be SCF_TMPL_VISIBILITY_HIDDEN,
SCF_TMPL_VISIBILITY_READONLY, or SCF_TMPL_VISIBILITY_READWRITE. If the visibility is unset, this function will return the default, SCF_TMPL_VISIBILITY_READWRITE.

The scf_tmpl_prop_cardinality() function will retrieve the minimum number of values and maximum number of values allowed for this property and place them in min and max, respectively. If the values are unset, the defaults of 0 for min and UINT64_MAX for max.

The scf_values_destroy() function destroys an scf_values_t structure and all memory associated with it.

The scf_values_t structure is populated by a number of functions. Based on the value type, it is populated with an array of the values. It is also always populated with an array of astring translations of those values.

typedef struct scf_time {
    int64_t t_seconds;
    int32_t t_ns;
} scf_time_t;

typedef struct scf_values {
    scf_type_t value_type;
    char *reserved;
    int value_count;
    const char **values_as_astring;
    union {
        uint64_t *v_count;
        uint8_t *v_boolean;
        int64_t *v_integer;
        char **v_astring;
        char **v_ustring;
        char **v_opaque;
        scf_time_t *v_time;
    } sv_data;
} scf_values_t;

The scf_tmpl_prop_internal_seps() function will retrieve the list of internal separators as currently defined in the template. Each separator will be a single string character in a different element of out. Some templates will not specify any internal separators. The caller is responsible for calling scf_values_destroy() on success.

The scf_tmpl_value_name_constraints() function will retrieve the set of property values the property is expected to be part of. Some templates will not specify any constraints. The caller is responsible for calling scf_values_destroy() on success.

The scf_tmpl_value_count_range_constraints() function will retrieve the set of defined lower and upper bounds as defined by the property template and place them in ranges. Some templates will not specify any range constraints.
The `scf_count_ranges_t` structure is populated by the `scf_tmpl_value_count_range_constraints()` and `scf_tmpl_value_count_range_choices()` functions. `scf_count_ranges_destroy()` destroys an `scf_count_ranges_t` and all memory associated with it.

```c
typedef struct scf_count_ranges {
    int scr_num_ranges;
    uint64_t *scr_min;
    uint64_t *scr_max;
} scf_count_ranges_t;
```

The `scf_tmpl_value_int_range_constraints()` function will retrieve the set of defined lower and upper bounds as defined by the property template and place them in `ranges`. Some templates will not specify any range constraints.

The `scf_int_ranges_t` structure is populated by the `scf_tmpl_value_int_range_constraints()` and `scf_tmpl_value_int_range_choices()` functions. The `scf_int_ranges_destroy()` function destroys an `scf_int_ranges_t` and all memory associated with it.

```c
typedef struct scf_int_ranges {
    int scr_num_ranges;
    int64_t *scr_min;
    int64_t *scr_max;
} scf_int_ranges_t;
```

The `scf_tmpl_value_name_choices()` function will retrieve the set of property value choices that should be offered to a user. Some templates will not specify any choices. The caller is responsible for calling `scf_values_destroy()` on success.

The `scf_tmpl_value_count_range_choices()` function will retrieve the set of defined lower and upper bounds as defined by the property template and place them in `ranges`. Some templates will not specify any range choices.

The `scf_tmpl_value_int_range_constraints()` function will retrieve the set of defined lower and upper bounds as defined by the property template and place them in `ranges`. Some templates will not specify any range constraints.

The `scf_tmpl_value_common_name()` function will retrieve the value's common name as currently templated and place it in `*out`. A locale (as described in `setlocale(3C)`) can be specified, or if the supplied locale is `NULL`, the current locale will be used. If a common name in the specified locale is not found, the function will also look for a common name in the C locale. Some templates will not specify the value common name. The caller is responsible for freeing the `*out` buffer on success.

The `scf_tmpl_value_description()` function will retrieve the value's description as currently templated and place it in `*out`. A locale (as described in `setlocale(3C)`) can be specified, or if the supplied locale is `NULL`, the current locale will be used. If a description in the
specified locale is not found, the function will also look for a description in the C locale. Some templates will not specify the value description. The caller is responsible for freeing the *out buffer on success.

The scf tmpl value_in_constraint() function will check that the value provided matches the constraints as defined in the property template provided. This currently means it will determine if the value provided:

- is of the proper type for the property template defined,
- is within a range defined, if it is a numeric type, and
- is within the name constraints, if name constraints are defined.

If the template property does not define a type, ranges will be considered of the same type as the numeric values being checked. Some ranges might consider the value out of constraint when tested as one numeric type but within constraint if tested as other numeric type. Refer to strtoull(3C) and strtoll(3C) to see the implications when retrieving numeric values from the repository or converting strings to numeric values in libscf(3LIB).

If errs is not NULL, an scf tmpl_error_t will be created, populated and added to errs in case of a constraint violation. The caller is responsible for calling scf tmpl_errors_destroy() to free memory allocated for all scf tmpl_error_t associated to errs.

**Return Values**

Upon successful completion, scf tmpl_prop_name(), scf tmpl_prop_common_name(), scf tmpl_prop_description(), scf tmpl_prop_units(), scf tmpl_value_common_name(), and scf tmpl_value_description() return the length of the string written, not including the terminating null byte. Otherwise, they return -1.

Upon successful completion, scf tmpl_prop_type(), scf tmpl_prop_required(), scf tmpl_prop_visibility(), scf tmpl_prop_cardinality(), scf tmpl_prop_internal_seps(), scf tmpl_value_name_constraints(), scf tmpl_value_count_range_constraints(), scf tmpl_value_int_range_constraints(), scf tmpl_value_name_choices(), scf tmpl_value_count_range_choices(), scf tmpl_value_int_range_choices() return 0. Otherwise, they return -1.

The scf tmpl value_in_constraint() functions returns 0 on success, 1 if the value is not in the constraint, and -1 on failure.

Upon successful completion, scf tmpl_visibility_to_string() returns a string of the visibility supplied.

**Errors**

The scf tmpl_prop_name(), scf tmpl_prop_type(), scf tmpl_prop_required(), scf tmpl_prop_common_name(), scf tmpl_prop_description(), scf tmpl_prop_units(), scf tmpl_prop_visibility(), scf tmpl_prop_cardinality(), scf tmpl_prop_internal_seps(), scf tmpl_value_name_constraints(), scf tmpl_value_count_range_constraints(), scf tmpl_value_int_range_constraints(), scf tmpl_value_name_choices(), scf tmpl_value_count_range_choices(), scf tmpl_value_int_range_choices(),
The `scf_tmpl_prop_name()` function will fail if:

**SCF_ERROR_TEMPLATE_INVALID** The template data is invalid.

The `scf_tmpl_prop_type()`, `scf_tmpl_prop_common_name()`,
`scf_tmpl_prop_description()`, `scf_tmpl_prop_units()`, `scf_tmpl_prop_cardinality()`,
`scf_tmpl_prop_internal_seps()`, `scf_tmpl_value_name_constraints()`,
`scf_tmpl_value_count_range_constraints()`, `scf_tmpl_value_int_range_constraints()`,
`scf_tmpl_value_name_choices()`, `scf_tmpl_value_count_range_choices()`,
`scf_tmpl_value_int_range_choices()`, `scf_tmpl_value_common_name()`, and `scf_tmpl_value_description()` functions will fail if:

**SCF_ERROR_NOT_FOUND** The property does not exist or exists and has no value.

The `scf_tmpl_value_in_constraint()` function will fail if:

**SCF_ERROR_INVALID_ARGUMENT** Value is not a valid `scf_value_t`.

The `scf_tmpl_prop_common_name()`, `scf_tmpl_prop_description()` and
`scf_tmpl_prop_units()` functions will fail if:

**SCF_ERROR_INVALID_ARGUMENT** The locale string is too long to make a property name.

The `scf_tmpl_value_common_name()` and `scf_tmpl_value_description()` functions will fail if:

**SCF_ERROR_BACKEND_ACCESS** The storage mechanism that the repository server
(svc.configd) chose for the operation denied access.

**SCF_ERROR_CONNECTION_BROKEN** The connection to the repository was lost.

**SCF_ERROR_DELETED** The template property group has been deleted.

**SCF_ERROR_HANDLE.Destroyed** The handle passed in has been destroyed.

**SCF_ERROR_INTERNAL** An internal error occurred.

**SCF_ERROR_NO_MEMORY** There is not enough memory to populate the
`scf_pg_tmpl_t`.

**SCF_ERROR_NO_RESOURCES** The server does not have adequate resources to complete
the request.

**SCF_ERROR_NOT_BOUND** The handle is not currently bound.

**SCF_ERROR_PERMISSION_DENIED** The template could not be read due to access
restrictions.

**SCF_ERROR_TEMPLATE_INVALID** The template data is invalid.
SCF_ERROR_INVALID_ARGUMENT The value and locale strings are too long to make a property name.

The scf_tmpl_value_count_range_constraints() and scf_tmpl_value_count_range_choices() functions will fail if:

SCF_ERROR_CONSTRAINT_VIOLATED The range has negative values.

The scf_tmpl_value_int_range_constraints() and scf_tmpl_value_int_range_choices() functions will fail if:

SCF_ERROR_CONSTRAINT_VIOLATED The range values don’t fit in a int64_t.

The scf_tmpl_value_count_range_constraints(), scf_tmpl_value_int_range_constraints(), scf_tmpl_value_count_range_choices() and scf_tmpl_value_int_range_choices() functions will fail if:

SCF_ERROR_CONSTRAINT_VIOLATED A range with min value > max value is found.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also svc.configd(1M), scf_tmpl_prop_create(3SCF), setlocale(3C), strtoll(3C), strtoull(3C), attributes(5), smf_template(5)
scf_tmpl_validate_fmri(3SCF)

Name scf_tmpl_validate_fmri, scf_tmpl_errors_destroy, scf_tmpl_next_error, scf_tmpl_reset_errors, scf_tmpl_strerror, scf_tmpl_error_type, scf_tmpl_error_source_fmri, scf_tmpl_error_pg_tmpl, scf_tmpl_error_pg, scf_tmpl_error_prop_tmpl, scf_tmpl_error_prop, scf_tmpl_error_value – template validation functions

Synopsis cc [ flag... ] file... -ls cf [ library... ]
#include <libscf.h>

int scf_tmpl_validate_fmri(scf_handle_t *h, const char *fmri, const char *snapshot, scf_tmpl_errors_t **errs, int flags);
void scf_tmpl_errors_destroy(scf_tmpl_errors_t *errs);
scf_tmpl_error_t *scf_tmpl_next_error(scf_tmpl_errors_t *errs, scf_tmpl_errors_t *err);
void scf_tmpl_reset_errors(scf_tmpl_errors_t *errs);
int scf_tmpl_strerror(scf_tmpl_error_t *err, char *s, size_t n, int flags);
int scf_tmpl_error_type(const scf_tmpl_error_t *err, scf_tmpl_error_type_t *type);
int scf_tmpl_error_source_fmri(const scf_tmpl_error_t *err, char *fmri);
int scf_tmpl_error_pg_tmpl(const scf_tmpl_error_t *err, char *name, char *type);
int scf_tmpl_error_pg(const scf_tmpl_error_t *err, char **name, char **type);
int scf_tmpl_error_prop_tmpl(const scf_tmpl_error_t *err, char **name, char **type);
int scf_tmpl_error_prop(const scf_tmpl_error_t *err, char **name, char **type);
int scf_tmpl_error_value(const scf_tmpl_error_t *err, char **val);

Description The template validation functions offer a way to validate the configuration data of a service instance against the appropriate template data. The scf_tmpl_validate_fmri() function returns the full set of errors for the specified instance, and those errors can be printed or explored directly.

By default, the validation is performed on the composed data from the running snapshot of an instance. A different snapshot can be explicitly selected by specifying a valid snapshot name rather than NULL for the snapshot argument. If flags includes SCF_TMPL_VALIDATE_FLAG_CURRENT, the snapshot argument is ignored and the current configuration is used.
By default, these functions also explore template data defined by the service or instance itself, the service's restarter, and global template data. See \texttt{smf_template(5)} for more information about this composition.

An instance FMRI is required, and FMRIs that specify other entities (for example, services) are explicitly rejected.

The \texttt{scf tmpl validate fmri()} function validates an instance FMRI against the template data in the repository. As described above, when the \texttt{snapshot} argument is NULL, the default running snapshot is used. If \texttt{scf tmpl errors t **} is non-null, the structure is allocated and returned to the caller for further perusal or printing of the errors.

The \texttt{scf tmpl errors destroy()} function destroys and frees the \texttt{scf tmpl errors t} and all of the \texttt{scf tmpl error t} structures to which it refers.

The \texttt{scf tmpl next error()} function takes a pointer to a \texttt{scf tmpl errors t} structure previously returned by \texttt{scf tmpl validate fmri()}. On the first call, it returns a pointer to the first \texttt{scf tmpl error t} found during validation. On subsequent calls, the next error is returned. To resume processing from the first error, the caller can use \texttt{scf tmpl reset errors()}.

The contents of an \texttt{scf tmpl error t} are determined by its type. Types added as additional validation checks are introduced. Based on the error type, a set of fields can be retrieved from the error.

\begin{verbatim}
SCF_TERR_TYPE_INVALID
  reserved invalid type

SCF_TERR_MISSING_PG
  required property group is missing

  template source FMRI
  property group template name and type

SCF_TERR_WRONG_PG_TYPE
  property group type is incorrect

  template source FMRI
  property group template name and type
  property group name and type

SCF_TERR_MISSING_PROP
  required property is missing

  template source FMRI
  property group template name and type

\end{verbatim}
property template name and type

**SCF.TERR.WRONG_PROP_TYPE**
property type is incorrect

template source FMRI
property group template name and type
property template name and type
property group name and type
property name and type

**SCF.TERR.CARDINALITY_VIOLATION**
number of values violates cardinality

template source FMRI
property group template name and type
property template name and type
property group name and type
property name and type
cardinality and cardinality limits

**SCF.TERR.VALUE_CONSTRAINT_VIOLATED**
constraint violated for value

template source FMRI
property group template name and type
property template name and type
property group name and type
property name and type
value

**SCF.TERR.RANGE_VIOLATION**
value violated specified range

template source FMRI
property group template name and type
property template name and type
property group name and type
property name and type
value

**SCF.TERR_PROP_TYPE_MISMATCH**
value type is different from property type
The SCF_TERR_PROP_TYPE_MISMATCH, SCF_TERR_VALUE_OUT_OF_RANGE and SCF_TERR_INVALID_VALUE types are only set from calls to scf_tmpl_value_in_constraint(3SCF).

The scf_tmpl_error_type() function retrieves the error type.

The scf_tmpl_error_source_fmri() function retrieves a string with the FMRI of the source of the template that was violated. This string is freed by scf_tmpl_errors_destroy().

The scf_tmpl_error_pg_tmpl() function retrieves strings with the name and type of the property group template that was violated. If the property group name or type was implicitly wildcarded (see smf_template(5)) in the template, this function returns a string containing SCF_TMPL_WILDCARD("*"). These strings are freed by scf_tmpl_errors_destroy().

The scf_tmpl_error_pg() function retrieves strings with the name and type of the property group that was violated. These strings are freed by scf_tmpl_errors_destroy().

The scf_tmpl_error_prop_tmpl() function retrieves strings with the name and type of the property template that was violated. If the property type was implicitly wildcarded (see smf_template(5)) in the template, this function returns a string containing SCF_TMPL_WILDCARD("*"). These strings are freed by scf_tmpl_errors_destroy().

The scf_tmpl_error_prop() function retrieves strings with the name and type of the property that was violated. These strings are freed by scf_tmpl_errors_destroy().

The scf_tmpl_error_value() function retrieves a string with the value containing the error in val. This string are freed by scf_tmpl_errors_destroy().
The `scf_tmpl_strerror()` function takes an `scf_tmpl_error_t` previously returned by `scf_tmpl_next_error()` and returns a `s`. If `flags` includes `SCF_TMPL_STRERROR_HUMAN`, `s` is a human-readable, localized description of the error. Otherwise, `s` is a one-line string suitable for logfile output.

**Return Values**

The `scf_tmpl_validate_fmri()` function returns 0 on successful completion with no validation failures. It returns 1 if there are validation failures. It returns -1 if there is an error validating the instance.

The `scf_tmpl_next_error()` function returns a pointer to the next `scf_tmpl_error_t`. When none remain, it returns `NULL`.

The `scf_tmpl_error_type()`, `scf_tmpl_error_source_fmri()`, `scf_tmpl_error_pg_tmpl()`, `scf_tmpl_error_pg()`, `scf_tmpl_error_prop_tmpl()`, `scf_tmpl_error_prop()`, and `scf_tmpl_error_value()` functions return 0 on success and -1 on failure.

The `scf_tmpl_strerror()` function returns the number of bytes that would have been written to `s` if `n` had been sufficiently large.

**Errors**

The `scf_tmpl_validate_fmri()` function will fail if:

- `SCF_ERROR_BACKEND_ACCESS`: The storage mechanism that the repository server (svc.configd(1M)) chose for the operation denied access.
- `SCF_ERROR_CONNECTION_BROKEN`: The connection to the repository was lost.
- `SCF_ERROR_DELETED`: The instance or one of its template property group have been deleted.
- `SCF_ERROR_HANDLE_DESTROYED`: The handle passed in has been destroyed.
- `SCF_ERROR_INTERNAL`: An internal error occurred.
- `SCF_ERROR_INVALID_ARGUMENT`: The handle argument, FMRI argument, or snapshot name is invalid.
- `SCF_ERROR_NO_MEMORY`: There is not enough memory to validate the instance.
- `SCF_ERROR_NO_RESOURCES`: The server does not have adequate resources to complete the request.
- `SCF_ERROR_NOT_BOUND`: The handle is not currently bound.
- `SCF_ERROR_NOT_FOUND`: An object matching FMRI does not exist in the repository, or the snapshot does not exist.
- `SCF_ERROR_PERMISSION_DENIED`: The instance or template could not be read due to access restrictions.
- `SCF_ERROR_TEMPLATE_INVALID`: The template data is invalid.
The `scf_tmpl_strerror()`, `scf_tmpl_error_type()`, `scf_tmpl_error_source_fmri()`, `scf_tmpl_error_pg_tmpl()`, `scf_tmpl_error_pg()`, `scf_tmpl_error_prop_tmpl()`, `scf_tmpl_error_prop()`, and `scf_tmpl_error_value()` functions will fail if:

**SCF_ERROR_INVALID_ARGUMENT**  
The `scf_tmpl_errors_t` argument is invalid.

The `scf_tmpl_error_type()`, `scf_tmpl_error_source_fmri()`, `scf_tmpl_error_pg_tmpl()`, `scf_tmpl_error_pg()`, `scf_tmpl_error_prop_tmpl()`, `scf_tmpl_error_prop()`, and `scf_tmpl_error_value()` functions will fail if:

**SCF_ERROR_NOT_FOUND**  
The data requested is not available for the `scf_tmpl_error_t` argument supplied.

**Attributes**  
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`sfc.configd(1M), scf_tmpl_value_in_constraint(3SCF), attributes(5), smf_template(5)`
Transactions are the mechanism for changing property groups. They act atomically, whereby either all of the updates occur or none of them do. An scf_transaction_t is always in one of the following states:

- **reset**: The initial state. A successful return of scf_transaction_start() moves the transaction to the started state.
- **started**: The transaction has started. The scf_transaction_property_delete(), scf_transaction_property_new(), scf_transaction_property_change(), and scf_transaction_property_change_type() functions can be used to set...
The `scf_transaction_reset()` and `scf_transaction_reset_all()` functions return the transaction to the reset state.

A call to `scf_transaction_commit()` (whether or not it is successful) moves the transaction to the committed state. Modifying, resetting, or destroying the entries and values associated with a transaction will move it to the invalid state.

The `scf_transaction_reset()` and `scf_transaction_reset_all()` functions return the transaction to the reset state.

The `scf_transaction_create()` function allocates and initializes an `scf_transaction_t` bound to `handle`. The `scf_transaction_destroy()` function resets, destroys, and frees `tran`. If there are any entries associated with the transaction, `scf_transaction_destroy()` also effects a call to `scf_transaction_reset()`. The `scf_transaction_destroy_children()` function resets, destroys, and frees all entries and values associated the transaction.

The `scf_transaction_handle()` function gets the handle to which `tran` is bound.

The `scf_transaction_start()` function sets up the transaction to modify the property group to which `pg` is set. The time reference used by `pg` becomes the basis of the transaction. The transaction fails if the property group has been modified since the last update of `pg` at the time when `scf_transaction_commit()` is called.

The `scf_transaction_property_delete()`, `scf_transaction_property_new()`, `scf_transaction_property_change()`, and `scf_transaction_property_change_type()` functions add a new transaction entry to the transaction. Each property the transaction affects must have a unique `scf_transaction_entry_t`. Each `scf_transaction_entry_t` can be associated with only a single transaction at a time. These functions all fail if the transaction is not in the started state, `prop_name` is not a valid property name, or `entry` is already associated with a transaction. These functions affect commit and failure as follows:

**scf_transaction_property_delete()**
This function deletes the property `prop_name` in the property group. It fails if `prop_name` does not name a property in the property group.

**scf_transaction_property_new()**
This function adds a new property `prop_name` to the property group with a value list of type `type`. It fails if `prop_name` names an existing property in the property group.

**scf_transaction_property_change()**
This function changes the value list for an existing property `prop_name` in the property group. It fails if `prop_name` does not name an existing property in the property group or names an existing property with a different type.
This function changes the value list and type for an existing property prop_name in the property group. It fails if prop_name does not name an existing property in the property group.

If the function call is successful, entry remains active in the transaction until scf_transaction_destroy(), scf_transaction_reset(), or scf_transaction_reset_all() is called. The scf_entry_add_value(3SCF) manual page provides information for setting up the value list for entries that are not associated with scf_transaction_property_delete(). Resetting or destroying an entry or value active in a transaction will move it into the invalid state.

The scf_transaction_commit() function attempts to commit tran.

The scf_transaction_reset() function returns the transaction to the reset state and releases all of the transaction entries that were added.

The scf_transaction_reset_all() function returns the transaction to the reset state, releases all of the transaction entries, and calls scf_value_reset(3SCF) on all values associated with the entries.

Upon successful completion, scf_transaction_create() returns a new scf_transaction_t. Otherwise, it returns NULL.

Upon successful completion, scf_transaction_handle() returns the handle associated with the transaction. Otherwise, it returns NULL.

Upon successful completion, scf_transaction_start(), scf_transaction_property_delete(), scf_transaction_property_new(), scf_transaction_property_change(), and scf_transaction_property_change_type() return 0. Otherwise, they return -1.

The scf_transaction_commit() function returns 1 upon successful commit, 0 if the property group set in scf_transaction_start() is not the most recent, and -1 on failure.

The scf_transaction_create() function will fail if:

- SCF_ERROR_INVALID_ARGUMENT The value of the handle argument is NULL.
- SCF_ERROR_NO_MEMORY There is not enough memory to allocate an scf_transaction_t.
- SCF_ERROR_NO_RESOURCES The server does not have adequate resources for a new transaction handle.

The scf_transaction_handle() function will fail if:

- SCF_ERROR_HANDLE_DESTROYED The handle associated with tran has been destroyed.

The scf_transaction_start() function will fail if:
SCF_ERROR_BACKEND_ACCESS  The repository backend refused the modification.
SCF_ERROR_BACKEND_READONLY  The repository backend refused modification because it is read-only.
SCF_ERROR_CONNECTION_BROKEN  The connection to the repository was lost.
SCF_ERROR_DELETED  The property group has been deleted.
SCF_ERROR_HANDLE_MISMATCH  The transaction and property group are not derived from the same handle.
SCF_ERROR_IN_USE  The transaction is not in the reset state. The scf_transaction_reset() and scf_transaction_reset_all() functions can be used to return the transaction to the reset state.
SCF_ERROR_NO_RESOURCES  The server does not have the resources to complete the request.
SCF_ERROR_NOT_BOUND  The handle was never bound or has been unbound.
SCF_ERROR_NOT_SET  The property group specified by $pg$ is not set.
SCF_ERROR_PERMISSION_DENIED  The user does not have sufficient privileges to modify the property group.

The scf_transaction_property_delete(), scf_transaction_property_new(), scf_transaction_property_change(), and scf_transaction_property_change_type() functions will fail if:

SCF_ERROR_BACKEND_ACCESS  The storage mechanism that the repository server (svc.configd(1M)) chose for the operation denied access.
SCF_ERROR_CONNECTION_BROKEN  The connection to the repository was lost.
SCF_ERROR_DELETED  The property group the transaction is changing has been deleted.
SCF_ERROR_HANDLE_MISMATCH  The transaction and entry are not derived from the same handle.
SCF_ERROR_IN_USE  The property already has an entry in the transaction.
SCF_ERROR_INTERNAL  An internal error occurred.
SCF_ERROR_INVALID_ARGUMENT  The $prop_name$ argument is not a valid property name.
SCF_ERROR_NO_RESOURCES  The server does not have the resources to complete the request.
SCF_ERROR_NOT_BOUND  The handle is not bound.
SCF_ERROR_NOT_SET  The transaction has not been started.
SCF_ERROR_TYPE_MISMATCH  The tran argument is not of a type compatible with type.

The scf_transaction_property_delete(), scf_transaction_property_change(), and scf_transaction_property_change_type() functions will fail if:
SCF_ERROR_EXISTS  The object already exists.
SCF_ERROR_NOT_FOUND  The property group does not contain a property named prop_name.

The scf_transaction_property_new(), scf_transaction_property_change(), and scf_transaction_property_change_type() functions will fail if:
SCF_ERROR_INVALID_ARGUMENT  The prop_name argument is not a valid property name, or the type argument is an invalid type.

The scf_transaction_property_new() function will fail if:
SCF_ERROR_EXISTS  The property group already contains a property named prop_name.
SCF_ERROR_NOT_FOUND  Nothing of that name was found.

The scf_transaction_property_change() function will fail if:
SCF_ERROR_TYPE_MISMATCH  The property prop_name is not of type type.

The scf_transaction_commit() function will fail if:
SCF_ERROR_BACKEND_READONLY  The repository backend is read-only.
SCF_ERROR_BACKEND_ACCESS  The repository backend refused the modification.
SCF_ERROR_NOT_BOUND  The handle is not bound.
SCF_ERROR_CONNECTION_BROKEN  The connection to the repository was lost.
SCF_ERROR_INVALID_ARGUMENT  The transaction is in an invalid state.
SCF_ERROR_DELETED  The property group the transaction is acting on has been deleted.
SCF_ERROR_NOT_SET  The transaction has not been started.
SCF_ERROR_PERMISSION_DENIED  The user does not have sufficient privileges to modify the property group.
SCF_ERROR_NO_RESOURCES  The server does not have sufficient resources to commit the transaction.

The scf_error(3SCF) function can be used to retrieve the error value.
EXAMPLE 1  Set an existing boolean value to true.

```c
    tx = scf_transaction_create(handle);
    e1 = scf_entry_create(handle);
    v1 = scf_value_create(handle);
    do {
        if (scf_pg_update(pg) == -1)
            goto fail;
        if (scf_transaction_start(tx, pg) == -1)
            goto fail;

        /* set up transaction entries */
        if (scf_transaction_property_change(tx, e1, "property",
                                               SCF_TYPE_BOOLEAN) == -1) {
            scf_transaction_reset(tx);
            goto fail;
        }
        scf_value_set_boolean(v1, 1);
        scf_entry_add_value(e1, v1);

        result = scf_transaction_commit(tx);
        scf_transaction_reset(tx);
    } while (result == 0);

    if (result < 0)
        goto fail;

    /* success */

    cleanup:
    scf_transaction_destroy(tx);
    scf_entry_destroy(e1);
    scf_value_destroy(v1);
```

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libscf(3LIB), scf_value_reset(3SCF), scf_error(3SCF), scf_pg_create(3SCF), attributes(5)
Name  scf_value_create, scf_value_handle, scf_value_reset, scf_value_destroy, scf_value_type,
        scf_value_base_type, scf_value_is_type, scf_type_base_type, scf_value_get_boolean,
        scf_value_get_count, scf_value_get_integer, scf_value_get_time, scf_value_get_astring,
        scf_value_get_ustring, scf_value_get_opaque, scf_value_get_as_string,
        scf_value_get_as_string_TYPED, scf_value_set_boolean, scf_value_set_count,
        scf_value_set_integer, scf_value_set_time, scf_value_set_from_string, scf_value_set_astring,
        scf_value_set_ustring, scf_value_set_opaque - manipulate values in the Service
        Configuration Facility

Synopsis  cc [ flag... ] file... -lscf [ library... ]

#include <libscf.h>

scf_value_t *scf_value_create(scf_handle_t *h);
scf_handle_t *scf_value_handle(scf_value_t *v);
void scf_value_reset(scf_value_t *v);
void scf_value_destroy(scf_value_t *v);
int scf_value_type(scf_value_t *v);
int scf_value_base_type(scf_value_t *v);
int scf_value_is_type(scf_value_t *v, scf_type_t type);
int scf_type_base_type(scf_type_t type, scf_type_t *out);
int scf_value_get_boolean(scf_value_t *v, uint8_t *out);
int scf_value_get_count(scf_value_t *v, uint64_t *out);
int scf_value_get_integer(scf_value_t *v, int64_t *out);
int scf_value_get_time(scf_value_t *v, int64_t *seconds,
                       int32_t *ns);
ssize_t scf_value_get_astring(scf_value_t *v, char *buf,
                               size_t size);
ssize_t scf_value_get_ustring(scf_value_t *v, char *buf,
                               size_t size);
ssize_t scf_value_get_opaque(scf_value_t *v, char *out,
                              size_t len);
ssize_t scf_value_get_as_string(scf_value_t *v, char *buf,
                                 size_t size);
ssize_t scf_value_get_as_string_TYPED(scf_value_t *v,
                                       scf_type_t type, char *buf, size_t size);
void scf_value_set_boolean(scf_value_t *v, uint8_t in);
void scf_value_set_count(scf_value_t *v, uint64_t in);
void scf_value_set_integer(scf_value_t *v, int64_t in);
The `scf_value_create()` function creates a new, reset `scf_value_t` that holds a single typed value. The value can be used only with the handle specified by `h` and objects associated with `h`.

The `scf_value_reset()` function resets the value to the uninitialized state. The `scf_value_destroy()` function deallocates the object.

The `scf_value_type()` function retrieves the type of the contents of `v`. The `scf_value_is_type()` function determines if a value is of a particular type or any of its subtypes. The `scf_type_base_type()` function returns the base type of `type`. The `scf_value_base_type()` function returns the true base type of the value (the highest type reachable from the value's type).

<table>
<thead>
<tr>
<th>Type Identifier</th>
<th>Base Type</th>
<th>Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCF_TYPE_INVALID</td>
<td></td>
<td>reserved invalid type</td>
</tr>
<tr>
<td>SCF_TYPE_BOOLEAN</td>
<td></td>
<td>single bit</td>
</tr>
<tr>
<td>SCF_TYPE_COUNT</td>
<td></td>
<td>unsigned 64-bit quantity</td>
</tr>
<tr>
<td>SCF_TYPE_INTEGER</td>
<td></td>
<td>signed 64-bit quantity</td>
</tr>
<tr>
<td>SCF_TYPE_TIME</td>
<td></td>
<td>signed 64-bit seconds, signed 32-bit nanoseconds in the range 0 (\leq) ns &lt; 1,000,000,000</td>
</tr>
<tr>
<td>SCF_TYPE_ASTRING</td>
<td>ASTRING</td>
<td>8-bit NUL-terminated string</td>
</tr>
<tr>
<td>SCF_TYPE_OPAQUE</td>
<td></td>
<td>opaque 8-bit data</td>
</tr>
<tr>
<td>SCF_TYPE_USTRING</td>
<td>USTRING</td>
<td>8-bit UTF-8 string</td>
</tr>
<tr>
<td>SCF_TYPE_URI</td>
<td>USTRING</td>
<td>a URI string</td>
</tr>
<tr>
<td>SCF_TYPE_FMRI</td>
<td>URI</td>
<td>a Fault Management Resource Identifier</td>
</tr>
<tr>
<td>SCF_TYPE_HOST</td>
<td>USTRING</td>
<td>either a hostname, IPv4 address, or IPv6 address</td>
</tr>
<tr>
<td>SCF_TYPE_HOSTNAME</td>
<td>HOST</td>
<td>a fully-qualified domain name</td>
</tr>
<tr>
<td>SCF_TYPE_NET_ADDR</td>
<td>HOST</td>
<td>a valid SCF_TYPE_NET_ADDR_V4 or SCF_TYPE_NET_ADDR_V6 address</td>
</tr>
</tbody>
</table>
The `scf_value_get_boolean()`, `scf_value_get_count()`, `scf_value_get_integer()`, `scf_value_get_time()`, `scf_value_get_astring()`, `scf_value_get_ustring()`, and `scf_value_get_opaque()` functions read a particular type of value from `v`.

The `scf_value_get_as_string()` and `scf_value_get_as_string_typed()` functions convert the value to a string form. For `scf_value_get_as_string_typed()`, the value must be a reachable subtype of `type`.

The `scf_value_set_boolean()`, `scf_value_set_count()`, `scf_value_set_integer()`, `scf_value_set_time()`, `scf_value_set_astring()`, `scf_value_set_ustring()`, and `scf_value_set_opaque()` functions set `v` to a particular value of a particular type.

The `scf_value_set_from_string()` function is the inverse of `scf_value_get_as_string()`. It sets `v` to the value encoded in `buf` of type `type`.

The `scf_value_set_*()` functions will succeed on `scf_value_t` objects that have already been set.

**Return Values**

Upon successful completion, `scf_value_create()` returns a new, reset `scf_value_t`. Otherwise, it returns NULL.

Upon successful completion, `scf_value_handle()` returns the handle associated with `v`. Otherwise, it returns NULL.

The `scf_value_base_type()` function returns the base type of the value, or `SCF_TYPE_INVALID` on failure.

Upon successful completion, `scf_value_type()` returns the type of the value. Otherwise, it returns `SCF_TYPE_INVALID`.

Upon successful completion, `scf_value_is_type()`, `scf_value_get_boolean()`, `scf_value_get_count()`, `scf_value_get_integer()`, `scf_value_get_time()`, `scf_value_set_time()`, `scf_value_set_from_string()`, `scf_value_set_astring()`, `scf_value_set_ustring()`, and `scf_value_set_opaque()` return 0. Otherwise, they return -1.

Upon successful completion, `scf_value_get_astring()`, `scf_value_get_ustring()`, `scf_value_get_as_string()`, and `scf_value_get_as_string_typed()` return the length of the source string, not including the terminating null byte. Otherwise, they return -1.

Upon successful completion, `scf_value_get_opaque()` returns the number of bytes written. Otherwise, it returns -1.
Errors

The `scf_value_create()` function will fail if:

- **SCF_ERROR_HANDLE_DESTROYED**
  The handle associated with `h` has been destroyed.

- **SCF_ERROR_INVALID_ARGUMENT**
  The handle is NULL.

- **SCF_ERROR_NO_MEMORY**
  There is not enough memory to allocate an `scf_value_t`.

The `scf_value_handle()` function will fail if:

- **SCF_ERROR_HANDLE_DESTROYED**
  The handle associated with `v` has been destroyed.

The `scf_value_set_time()` function will fail if:

- **SCF_ERROR_INVALID_ARGUMENT**
  The nanoseconds field is not in the range `0 <= ns < 1,000,000,000`.

The `scf_type_base_type()` function will fail if:

- **SCF_ERROR_INVALID_ARGUMENT**
  The `type` argument is not a valid type.

The `scf_value_set_astring()`, `scf_value_set_ustring()`, `scf_value_set_opaque()`, and `scf_value_set_from_string()` functions will fail if:

- **SCF_ERROR_INVALID_ARGUMENT**
  The `in` argument is not a valid value for the specified type or is longer than the maximum supported value length.

The `scf_type_base_type()`, `scf_value_is_type()`, and `scf_value_get_as_string_TYPED()` functions will fail if:

- **SCF_ERROR_INVALID_ARGUMENT**
  The `type` argument is not a valid type.

The `scf_value_type()`, `scf_value_base_type()`, `scf_value_get_boolean()`, `scf_value_get_count()`, `scf_value_get_integer()`, `scf_value_get_time()`, `scf_value_get_astring()`, `scf_value_get_ustring()`, `scf_value_get_as_string()`, and `scf_value_get_as_string_TYPED()` functions will fail if:

- **SCF_ERROR_NOT_SET**
  The `v` argument has not been set to a value.

The `scf_value_get_boolean()`, `scf_value_get_count()`, `scf_value_get_integer()`, `scf_value_get_time()`, `scf_value_get_astring()`, `scf_value_get_ustring()`, and `scf_value_get_as_string_TYPED()` functions will fail if:

- **SCF_ERROR_TYPE_MISMATCH**
  The requested type is not the same as the value's type and is not in the base-type chain.

The `scf_error(3SCF)` function can be used to retrieve the error value.

Attributes

See attributes(5) for descriptions of the following attributes:
scf_value_create(3SCF)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  
libscf(3LIB), scf_entry_add_value(3SCF), scf_error(3SCF), attributes(5)
**setflabel(3TSOL)**

### Name

setflabel – change effective sensitivity label of a file

### Synopsis

```
cc [flag...] file... -ltsol [library...]
#include <tsol/label.h>

int setflabel(const char *path, const m_label_t *label_p);
```

### Description

In most cases, the file that is named by `path` is relabeled by moving it to a new pathname relative to the root directory of the zone corresponding to `label_p`. If the source and destination file systems are loopback mounted from the same underlying file system, the file is renamed. Otherwise, the file is copied and removed from the source directory.

However, `setflabel()` behaves differently for files and directories which are on multilevel ZFS file systems. Refer to `zfs(1M)`. In that case, files are not moved, but are relabeled in place, since multilevel file systems support per-file labels.

The `setflabel()` function enforces the following policy checks:

- Files and directories on multilevel file systems are relabeled in place; they are not moved, and the relabel script described below does not apply to them.

  For multilevel file systems, the label of a directory cannot be changed if the directory is not empty. The new label for an object must dominate the label of its parent directory. If the new label does not match the label of the parent directory, the caller must have the `PRIV_FILE_UPGRADE_SL` privilege. The new label must be dominated by the `mlslabel` property of the file system. If the caller is not in the global zone, the zone label must dominate the new label.

  The remaining policy checks below apply to multilevel file systems as well, except where otherwise noted.

- If the sensitivity label of `label_p` equals the existing sensitivity label, then the file is not affected.

- If the corresponding directory does not exist in the destination zone, or if the directory exists, but has a different label than `label_p`, the file is not moved. Also, if the file already exists in the destination directory, the file is not moved. This does not apply to multilevel file systems.

- If the sensitivity label of the existing file is not equal to the calling process label and the caller is not in the global zone, then the file is not affected. If the caller is in the global zone and the file is not on a multilevel file systems, the existing file label must be in a labeled zone (not `ADMIN_LOW` or `ADMIN_HIGH`).

- If the calling process does not have write access to both the source and destination directories, then the calling process must have `PRIV_FILE_DAC_WRITE` in its set of effective privileges.
If the sensitivity label of $label_p$ provides read only access to the existing sensitivity label (an upgrade), then the user must have the `solaris.label.file.upgrade` authorization. In addition, if the current zone is a labeled zone, then it must have been assigned the privilege `PRIV_FILE_UPGRADE_SL` when the zone was configured.

If the sensitivity label of $label_p$ does not provide access to the existing sensitivity label (a downgrade), then the calling user must have the `solaris.label.file.downgrade` authorization. In addition, if the current zone is a labeled zone, then it must have been assigned the privilege `PRIV_FILE_DOWNGRADE_SL` when the zone was configured.

If the calling process is not in the global zone, and the user does not have the `solaris.label.range` authorization, then $label_p$ must be within the user's label range and within the system accreditation range.

If the existing file is in use (not tranquil) it is not affected. This tranquility check does not cover race conditions nor remote file access.

Additional policy constraints can be implemented by customizing the shell script `/etc/security/tsol/relabel`. See the comments in this file. Note that this script does not apply to multilevel file systems.

### Return Values

Upon successful completion, `setflabel()` returns 0. Otherwise it returns -1 and sets `errno` to indicate the error.

### Errors

The `setflabel()` function fails and the file is unchanged if:

- **EACCES**
  Search permission is denied for a component of the path prefix of `path`. The calling process does not have mandatory write access to the final component of `path` because the sensitivity label of the final component of `path` does not dominate the sensitivity label of the calling process and the calling process does not have `PRIV_FILE_MAC_WRITE` in its set of effective privileges.

- **EBUSY**
  There is an open file descriptor reference to the final component of `path`.

- **ECONNREFUSED**
  A connection to the label daemon could not be established.

- **EEXIST**
  A file with the same name exists in the destination directory.

- **EINVAL**
  Improper parameters were received by the label daemon.

  For callers not in the global zone and when `path` is not on a multilevel ZFS file system, the specified label does not match the caller's label.

  For multilevel ZFS file systems, the specified label is not dominated by all of the following: the file system `MLSLABEL` property, the label of the parent directory of `path`, and the caller's label.

- **EISDIR**
  The existing file is a directory.
ELOOP
Too many symbolic links were encountered in translating path.

EMLINK
The existing file is hardlinked to another file.

ENAMETOOLONG
The length of the path argument exceeds PATH_MAX.

ENOENT
The file referred to by path does not exist.

EROFS
The file system is read-only or its label is ADMIN_LOW or ADMIN_HIGH.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
zfs(1M), libtsol(3LIB), attributes(5)

“Setting a File Sensitivity Label” in Trusted Extensions Developer’s Guide

**Notes**
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
smf_enable_instance(3SCF)

Name

smf_enable_instance, smf_disable_instance, smf_refresh_instance, smf_restart_instance,
smf_maintain_instance, smf_degrade_instance, smf_restore_instance, smf_get_state,
smf_state_to_string, smf_state_from_string – administrative interface to the Service
Configuration Facility

Synopsis

cc [ flag... ] file... -lscf [ library... ]
#include <libscf.h>

int smf_enable_instance(const char * instance, int flags);
int smf_disable_instance(const char * instance, int flags);
int smf_refresh_instance(const char * instance);
int smf_restart_instance(const char * instance);
int smf_maintain_instance(const char * instance, int flags);
int smf_degrade_instance(const char * instance, int flags);
int smf_restore_instance(const char * instance);
int smf_set_restarter(const char * fmri, const char * restarter_fmri);
char *smf_get_state(const char * instance);
const char *smf_state_to_string(int32_t state_code);
int32_t smf_state_from_string(const char * state);

Description

These functions provide administrative control over service instances. Using these functions,
an administrative tool can make a request to enable, disable, refresh, or restart an instance. All
calls are asynchronous. They request an action, but do not wait to see if the action. They also
fail on incomplete instances. See scf_instance_is_complete(3SCF) for information about
incomplete instances.

The smf_enable_instance() function enables the service instance specified by instance
FMRI. If SMF_TEMPORARY is set in flags, the enabling of the service instance is a temporary
change, lasting only for the lifetime of the current system instance. The flags argument is set to
0 if no flags are to be used.

The smf_disable_instance() function places the service instance specified by instance
FMRI in the disabled state and triggers the stop method (see svc.startd(1M)). If
SMF_TEMPORARY is set in flags, the disabling of the service instance is a temporary change,
lasting only for the lifetime of the current system instance. The flags argument is set to 0 if no
flags are to be used.

The smf_refresh_instance() function causes the service instance specified by instance
FMRI to re-read its configuration information.

The smf_restart_instance() function restarts the service instance specified by instance
FMRI.
The `smf_set_restarter()` function sets the delegated restarter for the given instance. If the instance is online, the instance will be stopped before the change and restarted after the change is complete. If the `restarter_fnr` is NULL, then the restarter customization will be removed and set back to the file backed restarter. The master restarter `svc.startd` is defined with the following:

```c
#define SCF_INSTANCE_STARTD ((const char*)"svc:/system/svc/restarter:default")
```

The `smf_maintain_instance()` function moves the service instance specified by `instance` into the maintenance state. If `SMF_IMMEDIATE` is set in `flags`, the instance is moved into maintenance state immediately, killing any running methods. If `SMF_TEMPORARY` is set in `flags`, the change to maintenance state is a temporary change, lasting only for the lifetime of the current system instance. The `flags` argument is set to 0 if no flags are to be used.

The `smf_degrade_instance()` function moves an online service instance into the degraded state. This function operates only on instances in the online state. The `flags` argument is set to 0 if no flags are to be used. The only available flag is `SMF_IMMEDIATE`, which causes the instance to be moved into the degraded state immediately.

The `smf_restore_instance()` function brings an instance currently in the maintenance to the uninitialized state, so that it can be brought back online. For a service in the degraded state, `smf_restore_instance()` brings the specified instance back to the online state.

The `smf_get_state()` function returns a pointer to a string containing the name of the instance's current state. The user is responsible for freeing this string. Possible state strings are defined as the following:

```c
#define SCF_STATE_STRING_UNINIT ((const char*)"uninitialized")
#define SCF_STATE_STRING_MAINT ((const char*)"maintenance")
#define SCF_STATE_STRING_OFFLINE ((const char*)"offline")
#define SCF_STATE_STRING_DISABLED ((const char*)"disabled")
#define SCF_STATE_STRING_ONLINE ((const char*)"online")
#define SCF_STATE_STRING_DEGRADED ((const char*)"degraded")
```

The `smf_state_to_string()` function returns a pointer to an immutable string containing the state equivalent to `state_code`. Possible state strings are defined as above. Possible state codes are defined as following:

```c
#define SCF_STATE_UNINIT 0x00000001
#define SCF_STATE_MAINT 0x00000002
#define SCF_STATE_OFFLINE 0x00000004
#define SCF_STATE_DISABLED 0x00000008
#define SCF_STATE_ONLINE 0x00000010
#define SCF_STATE_DEGRADED 0x00000020
```

The `smf_state_from_string()` function returns the value equivalent to the string parameter state. Besides the strings defined above, this function accepts the string "all" as argument. In this case the returned value is:

```c
#define SCF_STATE_ALL 0x0000003F
```
Return Values

Upon successful completion, `smf_enable_instance()`, `smf_disable_instance()`, `smf_refresh_instance()`, `smf_restart_instance()`, `smf_maintain_instance()`, `smf_degrade_instance()`, `smf_set_restarter()`, and `smf_restore_instance()` return 0. Otherwise, they return -1.

Upon successful completion, `smf_get_state()` returns an allocated string. Otherwise, it returns NULL.

Upon successful completion `smf_state_to_string()` returns a pointer to a constant string. Otherwise, it returns NULL.

Upon successful completion `smf_state_from_string()` returns the macro value defined for the parameter state. Otherwise it returns -1.

Errors

These functions will fail if:

- `SCF_ERROR_NO_MEMORY`: The memory allocation failed.
- `SCF_ERROR_INVALID_ARGUMENT`: The instance FMRI or flags argument is invalid.
- `SCF_ERROR_NOT_FOUND`: The FMRI is valid but there is no matching complete instance found.
- `SCF_ERROR_CONNECTION_BROKEN`: The connection to repository was broken.
- `SCF_ERROR_NO_RESOURCES`: The server has insufficient resources.

The `smf_maintain_instance()`, `smf_refresh_instance()`, `smf_restart_instance()`, `smf_degrade_instance()`, `smf_set_restarter()`, and `smf_restore_instance()` functions will fail if:

- `SCF_ERROR_PERMISSION_DENIED`: User does not have proper authorizations. See `smf_security(5)`.
- `SCF_ERROR_BACKEND_ACCESS`: The repository's backend refused access.
- `SCF_ERROR_BACKEND_READONLY`: The repository's backend is read-only.

The `smf_restore_instance()` and `smf_degrade_instance()` functions will fail if:

- `SCF_ERROR_CONSTRAINT_VIOLATED`: The function is called on an instance in an inappropriate state.

The `smf_set_restarter()` function will fail if:

- `SCF_ERROR_CONSTRAINT_VIOLATED`: The function is called with a restarter fmri that is not an instance or is not online.

The `scf_error(3SCF)` function can be used to retrieve the error value.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  svc.startd(1M), libscf(3LIB), scf_error(3SCF), scf_instance_is_complete(3SCF), attributes(5), smf_security(5)
**Name**

smf_method_exit – exit from a Service Management Facility service method

**Synopsis**

```c
cc [ flag... ] file... -lscf [ library... ]
#include <libscf.h>

void smf_method_exit(int semantic, const char *message_short,
const char *message_long, const char *textdomain);
```

**Description**

The `smf_method_exit()` function calls `exit(2)` with `semantic`. The semantic should be one of the method exit codes defined in `smf_method(5)`.

If the calling process is a service method, then the remaining arguments will be made available to its rearter and should conform to the following guidelines:

- `message_short` should point to a null-terminated string with no spaces, containing a short, concise explanation of the reason for calling `smf_method_exit()`.
- `message_long` should point to a null-terminated string containing the reason for calling `smf_method_exit()`.
- `textdomain` should either be `NULL`, or point to a null-terminated string such that a call to `dgettext(3C)` with `textdomain` as `domainname` and `message_long` as `msgid` will return an appropriately localized target string.

See `svc.startd(1M)` or the manual page for the appropriate rearter for a description of semantic-specific behaviors. Use of `smf_method_exit()` with a rearter other than `svc.startd` or specifying a semantic not defined in `smf_method(5)` may produce unexpected results.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**

`svc.startd(1M), exit(2), dgettext(3C), libscf(3LIB), attributes(5), smf_method(5)`
**smf_notify_set_params**, **smf_notify_get_params**, **smf_notify_del_params** – store, retrieve and delete notification parameters in the Service Configuration Facility

### Synopsis

```c
#include <libscf.h>

int smf_notify_set_params(const char *class, nvlist_t *attr);
int smf_notify_get_params(nvlist_t **params, nvlist_t *nvl);
int smf_notify_del_params(const char *class, const char *fmri, int32_t tset);
```

### Parameters

- **class**
  - class of events to which the function call refers. For SMF state transition events, you should use a subclass of `SCF_SVC_TRANSITION_CLASS` defined in `<libscf.h>`.

- **tset**
  - set of SMF state transitions encoded in an `int32_t` value. The encoded states are defined in `<libscf.h>` by the macros `SCF_STATE_`.*. The encoding of an initial state is obtained by shifting 16 bits to the left the encoded value for that state. For example, the value of `tset` that represents the state transition set (see `smf(5)`, State Transition Sets) for to-maintenance, from-maintenance and form-online is given by:

  ```c
  set = ((SCF_STATE_MAINT | SCF_STATE_ONLINE) << 16) | SCF_STATE_MAINT;
  ```

- **fmri**
  - FMRI of an SMF service or instance

- **params**
  - address of a pointer to an `nvlist_t`

- **nvl**
  - a pointer to the event payload `nvlist_t`, such as is provided to an event delivery callback in `libfmevent(3LIB)`

- **attr**
  - a pointer to `nvlist_t` with the notification parameters

### Description

These interfaces are used to manipulate Notification Parameters for Software Events. See `smf(5)`

The **smf_notify_set_params()** function stores the notification parameters for **class**. The **attr nvlist_t** containing the notification parameters must follow the format:

```c
version (uint32_t)
fmri (string)
tset (int32_t)
SCF_NOTIFY_PARAMS (embedded nvlist)
  <mechanism-name> (embedded nvlist)
  <parameter-name> <parameter-type>
  ...
```
The version field must be SCF_NOTIFY_PARAMS_VERSION, defined in <libscf.h>. The fields fmri and tset are only required for SMF state transition events. They are ignored otherwise. Existing notification parameters are replaced by this function.

The smf_notify_get_params() function creates and populates the params nvlist_t with the notification parameters for the Event class in nvl. If the Event in nvl is an SMF state transition event, this function will perform a composed lookup in scf_instance_get_pg_composed(3SCF) for the instance FMRI in nvl. If notification parameters are not found in the composed lookup, the function will look for the system-wide notification parameters at SCF_INSTANCE_GLOBAL. The caller is responsible for calling nvlist_free(3NVPAIR) after using params. The params nvlist_t has the following format:

```
version (uint32_t)
SCF_NOTIFY_PARAMS (array of embedded nvlists)
  (start of notify-params[0])
    tset (int32_t)
      <mechanism-name> (embedded nvlist)
        <parameter-name> <parameter-type>
        ...
      (end <mechanism-name>)
    ...
  (end of notify-params[0])
...```

The SCF_NOTIFY_PARAMS is an array of nvlist_t because SMF state transitions have notification parameters for both end states of the transitions.

The smf_notify_del_params() function deletes the notification parameters for the given class. If class is not a subclass of SCF_SVC_TRANSITION_CLASS, fmri and tset are ignored.

Both smf_notify_del_params() and smf_notify_set_params() refresh all instances affected by the changes.

**Return Values**

Upon successful completion smf_notify_del_params(), smf_notify_get_params() and smf_notify_set_params() return SCF_SUCCESS. Otherwise they return SCF_FAILED.

**Errors**

These functions will fail if:

- **SCF_ERROR_BACKEND_ACCESS**
  The storage mechanism that the repository server (svc.configd(1M)) chose for the operation denied access.

- **SCF_ERROR_CONNECTION_BROKEN**
  The connection to the repository was lost.
SCF_ERROR_DELETED
    The entity being operated on has been deleted.

SCF_ERROR_INTERNAL
    An internal error occurred.

SCF_ERROR_INVALID_ARGUMENT
    An argument passed is invalid.

SCF_ERROR_NO_MEMORY
    There is not enough memory.

SCF_ERROR_NO_RESOURCES
    The server does not have the resources to complete the request.

SCF_ERROR_NOT_FOUND
    The entity was not found.

SCF_ERROR_PERMISSION_DENIED
    The caller does not have permission to access or modify the repository.

The smf_notify_del_params() and smf_notify_set_params() functions will fail if:

SCF_ERROR_BACKEND_READONLY
    The repository backend is read-only.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libfmevent(3LIB), libnvpair(3LIB), libscf(3LIB), nvlist_free(3NVPAIR), scf_error(3SCF), scf_instance_get_pg_composed(3SCF), attributes(5), smf(5)
**Name**  
srpt_SetDefaultState, srpt_GetDefaultState – set and retrieve the default state setting for the SRP Target service

**Synopsis**  
cc [ flag... ] file... -lsrpt [ library... ]  
#include <libsrpt.h>

```c
int srpt_SetDefaultState(boolean_t enabled);
int srpt_GetDefaultState(boolean_t *enabled);
```

**Parameters**  
`enabled`  boolean value indicating whether COMSTAR SRP targets should be created

**Description**  
The `srpt_SetDefaultState()` function sets the default behavior of the SRP Target service. If `enabled` is `B_TRUE`, SRP targets will be created for all discovered HCAs that have not been specifically disabled. If `enabled` is `B_FALSE`, targets will not be created unless the HCA has been specifically enabled. See `srpt_SetTargetState(3SRPT)` for enabling or disabling specific HCAs. If the default state is changed when the SRP service is online, the state of existing targets is not changed until the service is restarted.

The `srpt_GetDefaultState()` function returns the current value for `enabled`.

**Return Values**  
Upon successful completion, these functions return 0. Otherwise they return a non-zero value to indicate the error.

**Errors**  
These functions will fail if:

- `ENOMEM` Resources could not be allocated.
- `EINVAL` A parameter is invalid.

**Attributes**  
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`libsrt(3LIB), srpt_SetTargetState(3SRPT), attributes(5)`
srpt_SetTargetState (3SRPT)

**Name**
srpt_SetTargetState, srpt_GetTargetState, srpt_ResetTarget – set and retrieve SRP Target state for a specific HCA

**Synopsis**
```c
#include <libsrpt.h>

int srpt_SetTargetState(char *hca_guid, boolean_t enabled);
int srpt_GetTargetState(char *hca_guid, boolean_t *enabled);
int srpt_ResetTarget(char *hca_guid);
```

**Parameters**
- **hca_guid**: HCA GUID. Must be in one of the following forms:
  - `3BA000100CD18` base hex form
  - `0003BA000100CD18` base hex form with leading zeroes
  - `hca:3BA000100CD18` form from cfgadm
  - `eui.0003BA000100CD18` EUI form

- **enabled**: boolean value indicating whether a COMSTAR SRP target should be created for this HCA

**Description**
The `srpt_SetTargetState()` function controls whether a COMSTAR SRP target will be created for the specified HCA. If `enabled` is B_TRUE, an SRP target will be created for this HCA. If `enabled` is B_FALSE, a target will not be created. This function overrides the default setting for the SRP Target service as set by `srpt_SetDefaultState(3SRPT)`. Changing the target state takes effect immediately if the SRP target service is online. Targets set to disabled will be offline and removed; targets set to enabled will be immediately created.

The `srpt_GetTargetState()` function retrieves the current setting for the specified HCA.

The `srpt_ResetTarget()` function clears HCA-specific settings. The service-wide defaults will control SRP Target creation for this HCA.

**Return Values**
Upon successful completion, these functions return 0. Otherwise they return a non-zero value to indicate the error.

**Errors**
These functions will fail if:
- **ENOMEM**: Resources could not be allocated.
- **EINVAL**: A parameter is invalid.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
### srpt_SetTargetState(3SRPT)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: `libsrpt(3LIB), srpt_SetDefaultState(3SRPT), attributes(5)`
Name SSAAgentIsAlive, SSAGetTrapPort, SSARegSubtable, SSARegSubagent, SSARegSubtree, SSASendTrap, SSASubagentOpen – Sun Solstice Enterprise Agent registration and communication helper functions

Synopsis cc [ flag ... ] file ... -lssagent -lssasnmp [ library ... ]
# include <impl.h>

extern int SSAAgentIsAlive(IPAddress *agent_addr, int *port, char *community, struct timeval *timeout);
extern int SSAGetTrapPort();
extern int *SSARegSubagent(Agent *agent);
int SSARegSubtable(SSA_Table *table);
in SSARegSubtree(SSA_Subtree *subtree);
extern void SSASendTrap(char *name);
extern int SSASubagentOpen(int *num_of_retry, char *agent_name);

Description The SSAAgentIsAlive() function returns TRUE if the master agent is alive, otherwise returns FALSE. The agent_addr parameter is the address of the agent. Specify the security token in the community parameter. You can specify the maximum amount of time to wait for a response with the timeout parameter.

The SSAGetTrapPort() function returns the port number used by the Master Agent to communicate with the subagent.

The SSARegSubagent() function enables a subagent to register and unregister with a Master Agent. The agent parameter is a pointer to an Agent structure containing the following members:

```
int timeout; /* optional */
int agent_id; /* required */
int agent_status; /* required */
char *personal_file; /* optional */
char *config_file; /* optional */
char *executable; /* optional */
char *version_string; /* optional */
char *protocol; /* optional */
int process_id; /* optional */
char *name; /* optional */
int system_up_time; /* optional */
int watch_dog_time; /* optional */
Address address; /* required */
struct _Agent; /* reserved */
struct _Subtree; /* reserved */
```
The agent_id member is an integer value returned by the SSASubagentOpen() function. After calling SSASubagentOpen(), you pass the agent_id in the SSARegSubagent() call to register the subagent with the Master Agent.

The following values are supported for agent_status:

SSA_OPER_STATUS_ACTIVE
SSA_OPER_STATUS_NOT_IN_SERVICE
SSA_OPER_STATUS_DESTROY

You pass SSA_OPER_STATUS_DESTROY as the value in a SSARegSubagent() function call when you want to unregister the agent from the Master Agent.

Address has the same structure as sockaddr_in, that is a common UNIX structure containing the following members:

short sin_family;
ushort_t sin_port;
struct in_addr sin_addr;
char sin_zero[8];

The SSARegSubtable() function registers a MIB table with the Master Agent. If this function is successful, an index number is returned, otherwise 0 is returned. The table parameter is a pointer to a SSA_Table structure containing the following members:

int regTblIndex; /* index value */
int regTblAgentID; /* current agent ID */
Oid regTblOID; /* Object ID of the table */
int regTblStartColumn; /* start column index */
int regTblEndColumn; /* end column index */
int regTblStartRow; /* start row index */
int regTblEndRow; /* end row index */
int regTblStatus; /* status */

The regTblStatus can have one of the following values:

SSA_OPER_STATUS_ACTIVE
SSA_OPER_STATUS_NOT_IN_SERVICE

The SSARegSubtree() function registers a MIB subtree with the master agent. If successful this function returns an index number, otherwise 0 is returned. The subtree parameter is a pointer to a SSA_Subtree structure containing the following members:

int regTreeIndex; /* index value */
int regTreeAgentID; /* current agent ID */
Oid name; /* Object ID to register */
int regTreeStatus; /* status */

The regTreeStatus can have one of the following values:

SSA_OPER_STATUS_ACTIVE
SSA_OPER_STATUS_NOT_IN_SERVICE
The SSASendTrap() function instructs the Master Agent to send a trap notification, based on the keyword passed with name. When your subagent MIB is compiled by mibcodegen, it creates a lookup table of the trap notifications defined in the MIB. By passing the name of the trap notification type as name, the subagent instructs the Master Agent to construct the type of trap defined in the MIB.

The SSASubagentOpen() function initializes communication between the subagent and the Master Agent. You must call this function before calling SSARegSubagent() to register the subagent with the Master Agent. The SSASubagentOpen() function returns a unique agent ID that is passed in the SSARegSubagent() call to register the subagent. If 0 is returned as the agent ID, the attempt to initialize communication with the Master Agent was unsuccessful. Since UDP is used to initialize communication with the Master Agent, you may want to set the value of num_of_retry to make multiple attempts.

The value for agent_name must be unique within the domain for which the Master Agent is responsible.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
Name  SSAOidCmp, SSAOidCpy, SSAOidDup, SSAOidFree, SSAOidInit, SSAOidNew,
SSAOidString, SSAOidStrToOid, SSAOidZero – Sun Solstice Enterprise Agent OID helper functions

Synopsis  cc [ flag ... ] file ... -lssasnmp [ library ... ]
#include <impl.h>

int SSAOidCmp(Oid *oid1, Oid *oid2);
int SSAOidCpy(Oid *oid1, Oid *oid2, char *error_label);
Oid *SSAOidDup(Oid *oid, char *error_label);
void SSAOidFree(Oid *oid);
int SSAOidInit(Oid *oid, Subid *subids, int len, char *error_label);
Oid *SSAOidNew();
char *SSAOidString(Oid *oid);
Oid *SSAOidStrToOid(char* name, char *error_label);
void SSAOidZero(Oid *oid);

Description  The SSAOidCmp() function performs a comparison of the given OIDs. This function returns:
0   if oid1 is equal to oid2
1   if oid1 is greater than oid2
−1  if oid1 is less than oid2

The SSAOidCpy() function makes a deep copy of oid2 to oid1. This function assumes oid1 has
been processed by the SSAOidZero() function. Memory is allocated inside oid1 and the
contents of oid2, not just the pointer, is copied to oid1. If an error is encountered, an error
message is stored in the error_label buffer.

The SSAOidDup() function returns a clone of oid, by using the deep copy. Error information is
stored in the error_label buffer.

The SSAOidFree() function frees the OID instance, with its content.

The SSAOidNew() function returns a new OID.

The SSAOidInit() function copies the Subid array from subids to the OID instance with the
specified length len. This function assumes that the OID instance has been processed by the
SSAOidZero() function or no memory is allocated inside the OID instance. If an error is
encountered, an error message is stored in the error_label buffer.

The SSAOidString() function returns a char pointer for the printable form of the given oid.
The SSAoidStrToOid() function returns a new OID instance from \textit{name}. If an error is encountered, an error message is stored in the \textit{error\_label} buffer.

The SSAoidZero() function frees the memory used by the OID object for buffers, but not the OID instance itself.

\textbf{Return Values}  The SSAoidNew() and SSAoidStrToOid() functions return 0 if an error is detected.

\textbf{Attributes}  See \textit{attributes(5)} for descriptions of the following attributes:

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{ATTRIBUTE TYPE} & \textbf{ATTRIBUTE VALUE} \\
\hline
Interface Stability & Obsolete \\
MT\textcdot Level & Unsafe \\
\hline
\end{tabular}
\end{center}

\textbf{See Also}  \textit{attributes(5)}
Name | SSAStringCpy, SSAStringInit, SSAStringToChar, SSAStringZero – Sun Solstice Enterprise Agent string helper functions

Synopsis | cc [ flag ... ] file ... -lssasnmp [ library ... ]
#include <impl.h>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>void *SSAStringZero(String *string);</td>
<td>int SSAStringInit(String *string, uchar_t *chars, int len, char *error_label);</td>
</tr>
<tr>
<td>int SSAStringCpy(String *string1, String *string2, char *error_label);</td>
<td>char *SSAStringToChar(String string);</td>
</tr>
</tbody>
</table>

Description | The SSAStringCpy() function makes a deep copy of string2 to string1. This function assumes that string1 has been processed by the SSAStringZero() function. Memory is allocated inside the string1 and the contents of string2, not just the pointer, is copied to the string1. If an error is encountered, an error message is stored in the error_label buffer.

The SSAStringInit() function copies the char array from chars to the string instance with the specified length len. This function assumes that the string instance has been processed by the SSAStringZero() function or no memory is allocated inside the string instance. If an error is encountered, an error message is stored in the error_label buffer.

The SSAStringToChar() function returns a temporary char array buffer for printing purposes.

The SSAStringZero() function frees the memory inside of the String instance, but not the string object itself.

Return Values | The SSAStringInit() and SSAStringCpy() functions return 0 if successful and −1 if error.

Attributes | See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also | attributes(5)
**stmfAddToHostGroup(3STMF)**

**Name**  
stmfAddToHostGroup – add an initiator port to an existing host group

**Synopsis**  
`cc [ flag... ] file... -lstmf [ library... ]`  
#include `<libstmf.h>`

    int stmfAddToHostGroup(stmfGroupName *hostGroupName,
                          stmfDevid initiatorName);

**Parameters**
- **hostGroupName**  
The name of the host group to which the specified initiatorName is added.
- **initiatorName**  
The device identifier of the initiator port to add to the specified host group.

**Description**  
The stmfAddToHostGroup() function adds an initiator port to an existing host group.

**Return Values**  
The following values are returned:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMF_ERROR_EXISTS</td>
<td>The specified initiatorName already exists in this hostGroupName or in another host group in the system.</td>
</tr>
<tr>
<td>STMF_ERROR_GROUP_NOT_FOUND</td>
<td>The specified hostGroupName was not found in the system.</td>
</tr>
<tr>
<td>STMF_STATUS_SUCCESS</td>
<td>The API call was successful.</td>
</tr>
</tbody>
</table>

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
libstmf(3LIB), attributes(5)
Name: stmfAddToHostGroupList – add an initiator port to a list of existing host groups

Synopsis: cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfAddToHostGroupList(stmfGroupList *hostGroupList,
                            stmfDevid *initiatorName, void **errData);

Parameters:
- **hostGroupList**: The host group list to which the specified **initiatorName** is to be added.
- **initiatorName**: The device identifier of the initiator port to be added to the specified host group.
- **errData**: A pointer to a pointer to void. On failure, this may contain appropriate information depending on the error value returned. For some types of failure it is set to NULL.

Description: The stmfAddToHostGroupList() function adds an initiator port to a list of host groups. The host groups mentioned in the list should already exist in the system. If any one of the given host groups in the list fails the system validity checks or if adding the specified initiator to any of the host groups causes conflict, the initiator will not be added to any of the host groups. Depending on the type of error, the parameter **errData** may get populated with the details of the error that caused the failure. The caller should call stmfFreeMemory(3STMF) when that data are no longer needed.

Return Values: The following values are returned:

- **STMF_ERROR_MEMBER_NOT_FOUND**: The specified **initiatorName** is invalid.
- **STMF_ERROR_GROUP_NOT_FOUND**: One or more specified host group names mentioned in the **hostGroupList** is invalid. This failure also populates **errData** with a pointer to a pointer to a stmfGroupList structure with the invalid host groups information.
- **STMF_ERROR_EXISTS**: One or more specified host group names mentioned in the **hostGroupList** is not found in the system. This failure also populates **errData** with a pointer to a pointer to a stmfGroupList structure with information about the host groups that caused the error.
- **STMF_ERROR_VE_CONFLICT**: The specified **initiatorName** could not be added to the **hostGroupsList** because such addition would cause unresolvable conflicts with existing view entries in the system. This failure also populates **errData** with a pointer to a pointer to a stmfViewList structure with a list of views that caused the conflict.
- **STMF_STATUS_SUCCESS**: The API call was successful.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libstm(3LIB), stmfFreeMemory(3STMF), attributes(5)
stmfAddToTargetGroup(3STMF)

**Name**
stmfAddToTargetGroup – add a target to an existing target group

**Synopsis**
```c
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfAddToTargetGroup(stmfGroupName *targetGroupName,
                          stmfDevid targetName);
```

**Parameters**
- **targetGroupName** The name of the target port group to which the specified `targetName` is added.
- **targetName** The device identifier of the target port to add to the specified target group.

**Description**
The `stmfAddToTargetGroup()` function adds a target to an existing target group.

**Return Values**
The following values are returned:
- **STMF_ERROR_EXISTS** The specified `targetName` already exists in this `targetGroupName` or in another target group in the system.
- **STMF_ERROR_GROUP_NOT_FOUND** The specified `targetGroupName` was not found in the system.
- **STMF_STATUS_SUCCESS** The API call was successful.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also** libstmf(3LIB), attributes(5)
stmfAddViewEntry - add a view entry for a given logical unit

Synopsis
cc [ flag... ] file... - STLmF [ library... ]
#include <libstmf.h>

int stmfAddViewEntry(stmfGuid *logicalUnit,
                      stmfViewEntry *viewEntry);

Parameters
    logicalUnit       The identifier of the logical unit to which this view entry is being added.
    viewEntry         The view entry to add to the specified logical unit identifier.

Description
    The stmfAddViewEntry() function adds a view entry for a given logical unit.

Return Values
    The following values are returned:
    STMF_ERROR_LUN_IN_USE   The specified logical unit number is already in use for this logical unit.
    STMF_ERROR_NOT_FOUND    The ID specified for logicalUnit was not found in the system.
    STMF_ERROR_VE_CONFLICT  Adding this view entry is in conflict with one or more existing view entries.
    STMF_STATUS_SUCCESS     The API call was successful.

Attributes
    See attributes for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also
    libstmf(3LIB), attributes

Notes
    If luNbrValid in the stmfViewEntry structure is set to B_FALSE, the framework will assign a logical unit number for this view entry. veIndexValid must be set to B_FALSE when adding a view entry. On successful return, veIndexValid will be set to B_TRUE and veIndex will contain the view entry index assigned to this view entry by the framework.
stmfAddViewEntryList– add a list of view entries for a given logical unit

Synopsis

cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfAddViewEntryList(stmfGuid *logicalUnit,
                          stmfViewEntryList *viewEntryList, void **opData);

Parameters

- **logicalUnit**: The identifier of the logical unit to which all view entries in the list are being added.
- **viewEntryList**: The list of view entries to be added to the specified logical unit identifier.
- **opData**: A pointer to a pointer to void. On successful return, *opData* will contain the information about the list of view entries added. On failure, *opData* may contain appropriate information depending on the error value returned. For some types of failure it is set to NULL.

Description

The *stmfAddViewEntryList()* function adds all view entries mentioned in the list for the specified logical unit. Details of the view entries to be added are passed to the function using the *stmfViewEntry* structure.

If the caller set the field *luNbrValid* to *B_FALSE*, the system will automatically choose a logical unit number (LUN) per initiator associated with host group mentioned in the view entry. Otherwise, the caller can specify a fixed LUN using the field *luNbr* along with the field *luNbrValid* set to *B_TRUE*. In this case, the system will map the specified LUN to the given logical unit for all initiators associated with the host group mentioned in the view entry, if the given LUN does not conflict with any other fixed LUN already existing in the system for any one of the above mentioned initiators. If the specified fixed LUN is in conflict with another system-chosen automatic LUN, the system-chosen automatic LUN will be remapped to another number to avoid the conflict.

If any one of the given view entries in the list fails the system validity checks or causes conflicts, none of the specified view entries will be added to the system. The *opData* parameter will be populated with the details which caused the failure. The information in *opData* needs to be interpreted depending on the return value of the function. If the function call succeeds, *opData* will be populated with information regarding newly added view entries. Some of the failure types may set this parameter to NULL. If the library returns data through parameter *opData*, the caller should call *stmfFreeMemory(3STMF)* when that data are no longer needed.

Return Values

The following values are returned:

- **STMF_STATUS_SUCCESS**: The API call was successful. This failure also sets *opData* to a pointer to a pointer to a *stmfViewEntryList* structure with the information about view entries added to the specified logical unit.
STMF_ERROR_INVALID_TG
One or more target group names mentioned in the view entry structure are invalid. This failure sets opData to a pointer to a pointer to a stmfViewList structure with view entries that caused the error.

STMF_ERROR_INVALID_HG
One or more host group name mentioned in the view entry structure are invalid. This failure sets opData to a pointer to a pointer to a stmfViewList structure with view entries that caused the error.

STMF_ERROR_INVALID_ARG
One or more view entries has an invalid value. Examples for invalid view entry values are providing a bad luNbr field with the luNbrValid field set to TRUE or providing a bad or used velIndex value with the velIndexValid field set to TRUE. This failure sets opData to a pointer to a pointer to a stmfViewList structure with view entries that caused the error.

STMF_ERROR_EXISTS
One or more view entries mentioned in the viewEntryList already exist in the system. This failure sets opData to a pointer to a pointer to a stmfViewList structure with view entries that caused the error.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also: libstmf(3LIB), stmfFreeMemory(3STMF), attributes(5)
stmfCheckHostGroupInUse – check if a particular host group is in use

Synopsis

```
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfCheckHostGroupInUse(stmfGroupName *groupName, boolean_t *inUse);
```

Parameters

- `groupName`: The name of the host group being checked.
- `inUse`: `B_TRUE` if the group is in use, `B_FALSE` if the group is not in use.

Description

The `stmfCheckHostGroupInUse()` function checks whether a given host group is in use. A host group is considered to be in use when there are one or more view entries using the host group.

Return Values

The following values are returned:

- `STMF_STATUS_SUCCESS`: The API call was successful.
- `STMF_ERROR_INVALID_ARG`: The `groupName` argument was unrecognized.
- `STMF_ERROR_NOT_FOUND`: The specified `groupName` was not found in the system.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also

`libstmf(3LIB), attributes(5)`
stmfCheckTargetGroupInUse(3STMF)

<table>
<thead>
<tr>
<th>Name</th>
<th>stmfCheckTargetGroupInUse – check if a particular target group is in use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>cc [ flag... ] file... -lstmf [ library... ]</code></td>
</tr>
<tr>
<td></td>
<td><code>#include &lt;libstmf.h&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>int stmfCheckTargetGroupInUse(stmfGroupName *groupName, boolean_t *inUse);</code></td>
</tr>
<tr>
<td>Parameters</td>
<td><strong>groupName</strong>  The name of the target group being checked.</td>
</tr>
<tr>
<td></td>
<td><strong>inUse</strong>      B_TRUE if the group is in use. B_FALSE if the group is not in use.</td>
</tr>
<tr>
<td>Description</td>
<td>The stmfCheckTargetGroupInUse() function checks whether a target group is in use. A target group is considered to be in use when there are one or more view entries using the target group.</td>
</tr>
<tr>
<td>Return Values</td>
<td>The following values are returned:</td>
</tr>
<tr>
<td></td>
<td><strong>STMF_STATUS_SUCCESS</strong> The API call was successful.</td>
</tr>
<tr>
<td></td>
<td><strong>STMF_ERROR_INVALID_ARG</strong> The <strong>groupName</strong> argument was unrecognized.</td>
</tr>
<tr>
<td></td>
<td><strong>STMF_ERROR_NOT_FOUND</strong> The specified <strong>groupName</strong> was not found in the system.</td>
</tr>
<tr>
<td>Attributes</td>
<td>See attributes(5) for descriptions of the following attributes:</td>
</tr>
<tr>
<td></td>
<td><strong>ATTRIBUTE TYPE</strong></td>
</tr>
<tr>
<td></td>
<td>Interface Stability</td>
</tr>
<tr>
<td></td>
<td>MT-Level</td>
</tr>
<tr>
<td>See Also</td>
<td>libstmf(3LIB), attributes(5)</td>
</tr>
</tbody>
</table>
stmfClearProviderData – delete all data for the specified provider

**Synopsis**

```c
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfClearProviderData(char *providerName, int providerType);
```

**Parameters**

- `providerName` The name of the provider whose data is being deleted.
- `providerType` The value must be either `STMF_LU_PROVIDER_TYPE` or `STMF_PORT_PROVIDER_TYPE`.

**Description**

The `stmfClearProviderData()` function deletes all data for the specified provider.

**Return Values**

The following values are returned:

- `STMF_ERROR_NOT_FOUND` The value specified for `providerName` was not found in the system.
- `STMF_STATUS_SUCCESS` The API call was successful.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**

`libstmf(3LIB), attributes(5)`
Name  stmfCreateHostGroup – create a new host group

Synopsis  cc [ flag... ] file... -lstmf [ library... ]
          #include <libstmf.h>

          int stmfCreateHostGroup(stmfGroupName *hostGroupName);

Parameters  hostGroupName  The name of the host group to be created.

Description  The stmfCreateHostGroup() function creates a new host group.

Return Values  The following values are returned:

          STMF_ERROR_EXISTS  The value specified for hostGroupName already exists in the system.
          STMF_INVALID_ARGUMENT  The value specified for hostGroupName was not valid.
          STMF_STATUS_SUCCESS  The API call was successful.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libstmf(3LIB), attributes(5)
stmfCreateLu – create a logical unit

Synopsis

```c
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfCreateLu(luResource hdl, stmfGuid *luGuid);
```

Parameters

- `hdl` The logical unit resource returned from a previous call to `stmfCreateLuResource(3STMF)`.
- `luGuid` If non-null, it must contain a pointer to an `stmfGuid` structure allocated by the caller. On successful return from this API, it will contain the guid of the newly created logical unit. If `luGuid` is NULL, this argument is ignored.

Description

The `stmfCreateLu` function creates a logical unit in stmfs using the properties of `hdl`. See `stmfSetLuProp(3STMF)` for a complete description of properties and their possible values.

Return Values

The following values are returned:

- `STMF_STATUS_SUCCESS` The API call was successful.
- `STMF_ERROR_FILE_IN_USE` The filename specified by the `STMF_LU_PROP_DATA_FILENAME` or `STMF_LU_PROP_META_FILENAME` was in use.
- `STMF_ERROR_GUID_IN_USE` The guid specified by the `STMF_LU_PROP_GUID` property is already being used.
- `STMF_ERROR_INVALID_BLKSIZE` The blocksize specified by `STMF_LU_PROP_BLOCK_SIZE` is invalid.
- `STMF_ERROR_WRITE_CACHE_SET` The requested write cache setting could not be provided.
- `STMF_ERROR_SIZE_OUT_OF_RANGE` The specified logical unit size is not supported.
- `STMF_ERROR_META_FILE_NAME` The specified meta file could not be accessed.
- `STMF_ERROR_DATA_FILE_NAME` The specified data file could not be accessed.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>
See Also  libstmf(3LIB), stmfCreateLuResource(3STMF), stmfSetLuProp(3STMF), attributes(5)
stmfCreateLuResource – create new logical unit resource

**Synopsis**
```
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfCreateLuResource(uint16_t dType, luResource *hdl);
```

**Parameters**
- `dType` The device type of the logical unit resource. Only `STMF_DISK` is currently supported.
- `hdl` The logical unit resource to be created.

**Description**
The `stmfCreateLuResource()` function creates a resource for setting properties of a logical unit for purposes of creating a logical unit in STMF.

**Return Values**
The following values are returned:
- `STMF_ERROR_INVALID_ARG` Either type is unrecognized or `hdl` was NULL.
- `STMF_STATUS_SUCCESS` The API call was successful.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `libstmf(3LIB)`
- attributes(5)
stmfCreateTargetGroup – create a new target port group

Synopsis

cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfCreateTargetGroup(stmfGroupName *targetGroupName);

Parameters

targetGroupName  The name of the target port group to be created.

Description

The stmfCreateTargetGroup() function creates a new target port group.

Return Values

The following values are returned:

- **STMF_ERROR_EXISTS**: The value specified for targetGroupName already exists in the system.
- **STMF_INVALID_ARGUMENT**: The value specified for targetGroupName was not valid.
- **STMF_STATUS_SUCCESS**: The API call was successful.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also

libstmf(3LIB), attributes(5)
stmfDeleteHostGroup(3STMF)

**Name**
stmfDeleteHostGroup – delete an existing host group

**Synopsis**
```
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfDeleteHostGroup(stmfGroupName *hostGroupName);
```

**Parameters**
*hostGroupName*  The name of the host group being deleted.

**Description**
The stmfDeleteHostGroup() function deletes an existing host group.

**Return Values**
The following values are returned:

- **STMF_ERROR_NOT_FOUND**: The specified *hostGroupName* was not found in the system.
- **STMF_STATUS_SUCCESS**: The API call was successful.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**
libstmf(3LIB), attributes(5)
stmfDeleteLu - delete a logical unit

Synopsis

cc [ flag... ] file... -Istmf [ library... ]
#include <libstmf.h>

int stmfDeleteLu(stmfGuid *luGuid);

Parameters

luGuid  a pointer to an stmfGuid structure containing the guid of the logical unit to delete

Description

The stmfDeleteLu() function deletes the logical unit from the system. Any view entries that may exist for this logical unit will be retained in the system and must be removed using stmfRemoveViewEntry(3STMF) if so desired.

Return Values

The following values are returned:

STMF_STATUS_SUCCESS  The API call was successful.
STMF_ERROR_NOT_FOUND  The guid does not exist.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also

libstmf(3LIB), stmfRemoveViewEntry(3STMF), attributes(5)
stmfDeleteTargetGroup(3STMF)

Name  
stmfDeleteTargetGroup – delete an existing target port group

Synopsis  
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfDeleteTargetGroup(stmfGroupName *targetGroupName);

Parameters  
targetGroupName  The name of the target port group being deleted.

Description  
The stmfDeleteTargetGroup() function deletes an existing target port group.

Return Values  
The following values are returned:

- STMF_ERROR_NOT_FOUND  The specified targetGroupName was not found in the system.
- STMF_STATUS_SUCCESS  The API call was successful.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  
libstmf(3LIB), attributes(5)
Name          stmfDestroyProxyDoor – close the door interface

Synopsis      cc [ flag... ] file... - lstmf [ library... ]
              #include <libstmf.h>

              void stmfDestroyProxyDoor(int hdl);

Parameters     hdl          handle returned from a previous call to stmfInitProxyDoor(3STMF)

Description   The stmfDestroyProxyDoor() function closes the door interface established in the call to stmfInitProxyDoor().

Attributes     See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also         libstmf(3LIB), stmfInitProxyDoor(3STMF), attributes(5)
stmfDevidFromIscsiName(3STMF)

Name
stmfDevidFromIscsiName – convert an iSCSI name to a stmfDevid structure

Synopsis
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfDevidFromIscsiName(char *iscsiName, stmfDevid *devid);

Parameters
iscsiName A character string of UTF-8 encoded Unicode characters representing the
iSCSI name terminated with the Unicode nul character.
devid A pointer to a stmfDevid structure allocated by the caller. On successful return,
this will contain the converted device identifier. On error, the value of this
parameter is undefined.

Description
The stmfDevidFromIscsiName() function converts an iSCSI name to a stmfDevid structure.
It returns the devid as a SCSI name string identifier.

Return Values
The following values are returned:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMF_ERROR_INVALID_ARGUMENT</td>
<td>The value of iscsiName was not valid iSCSI name.</td>
</tr>
<tr>
<td>STMF_STATUS_SUCCESS</td>
<td>The API call was successful.</td>
</tr>
</tbody>
</table>

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also
libstmf(3LIB), attributes(5)
**Name**  
stmfDevidFromWwn – convert a WWN to a stmfDevid structure

**Synopsis**  
cc [ flag... ] file... -lstmf [ library... ]  
#include <libstmf.h>

```c
int stmfDevidFromWN(uchar_t wnn[8], stmfDevid *devid);
```

**Parameters**  
*wwn*  
The 8-byte WWN identifier.

*devid*  
A pointer to a stmfDevid structure allocated by the caller. On successful return, this will contain the converted device identifier. On error, the value of this parameter is undefined.

**Description**  
The stmfDevidFromWwn function convert a WWN to a stmfDevid structure. It returns the *devid* as a SCSI name string.

**Return Values**  
The following values are returned:

- **STMF_ERROR_INVALID_ARGUMENT**  
The value of *wwn* was not valid WWN identifier.

- **STMF_STATUS_SUCCESS**  
The API call was successful.

**Attributes**  
See [attributes(5)] for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
libstmf(3LIB), attributes(5)
stmfFreeLuResource(3STMF)

Name
stmfFreeLuResource – free an allocated logical unit resource

Synopsis
cc [ flag... ] file... -lstmf [ library... ]
#include "libstmf.h"

int stmfFreeLuResource(luResource hdl);

Parameters
hdl A logical unit resource previously allocated in a call to
stmfCreateLuResource(3STMF) or stmfGetLuResource(3STMF).

Description
The stmfFreeLuResource() function frees a logical unit resource that was previously
allocated in a call to stmfCreateLuResource(3STMF) or stmfGetLuResource(3STMF).

Return Values
The following values are returned:

STMF_STATUS_SUCCESS The API call was successful.
STMF_ERROR_INVALID_ARG The hdl argument is not a valid logical unit resource.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also
libstm(3LIB), stmfCreateLuResource(3STMF), stmfGetLuResource(3STMF),
attributes(5)
stmfFreeMemory

**Name**  
stmfFreeMemory – free memory allocated by this library

**Synopsis**  
c

```c
#include <libstmf.h>

void stmfFreeMemory(void *stmfMemory);
```

**Parameters**  
memory  
A pointer to memory that was previously allocated by this library. If stmfMemory() is equal to NULL, the call will return successfully.

**Description**  
The stmfFreeMemory() function frees memory allocated by this library.

**Return Values**  
No values are returned.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
libstmf(3LIB), attributes(5)
Name  stmfFreeViewResourceList – free allocated list of view resources

Synopsis  

cc [ flag... ] file... -lstmf [ library... ]

#include <libstmf.h>

int stmfFreeViewResourceList(viewResourceList *resourceList);

Parameters  

resourceList  A list of view resources previously allocated using stmfGetViewResourceList(3STMF) call.

Description  The stmfFreeViewResourceList() function frees the list of view resources that was previously allocated using stmfGetViewResourceList(3STMF) call.

Return Values  The following values are returned:

STMF_STATUS_SUCCESS
  The API call was successful.

STMF_ERROR_INVALID_ARG
  The passed argument is not valid.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libstmf(3LIB), stmfGetViewResourceList(3STMF), attributes(5)
StmfGetAluaState – return the Asymmetric Logical Unit Access State (ALUA) mode for STMF

Synopsis

```
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetAluaState(boolean_t *alua_enabled, uint32_t *node);
```

Parameters

- `alua_enabled`: Set to `B_TRUE` or `B_FALSE` on success.
- `node`: Set to 0 or 1 on success.

Description

The `stmfGetAluaState()` function returns the Asymmetric Logical Unit Access State (ALUA) mode for STMF along with the node setting.

Return Values

The following values are returned:

- `STMF_ERROR_INVALID_ARG`: Either `alua_enabled` or `node` was incorrectly set.
- `STMF_STATUS_SUCCESS`: The API call was successful.

Attributes

See `attributes(5)` for descriptions of the following attributes:

```
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>
```

See Also

- `libstmf(3LIB)`, `attributes(5)`
stmfGetHostGroupList(3STMF)

Name
stmfGetHostGroupList – retrieve the list of host groups

Synopsis
cc [ flag... ] file... -lstmf [ library... ]  
#include <libstmf.h>

int stmfGetInitiatorGroupList(stmfGroupList **hostGroupList);

Parameters
hostGroupList  A pointer to a pointer to an stmfGroupList structure. On successful return, 
this will contain a list of host groups.

Description
The stmfGetInitiatorGroupList() function retrieves the list of host groups. The caller 
should call stmfFreeMemory(3STMF) when this list is no longer needed.

Return Values
The following values are returned:

- STMF_ERROR_NOMEM  The library was unable to allocate sufficient memory for 
  hostGroupList.
- STMF_STATUS_SUCCESS  The API call was successful.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also
stmfFreeMemory(3STMF), libstmf(3LIB), attributes(5)
**stmfGetHostGroupMembers (3STMF)**

**Name**  
stmfGetHostGroupMembers – retrieve the properties of the specified host group

**Synopsis**  
cc [ flag... ] file... -lstmf [ library... ]  
#include <libstmf.h>

```c
int stmfGetHostGroupMembers(stmfGroupName *hostGroupName,
                            stmfGroupProperties **groupProperties);
```

**Parameters**  

- **hostGroupName**  
  The name of the host group whose member list is being retrieved.

- **groupProperties**  
  A pointer to a pointer to an stmfGroupProperties structure. On successful return, this will contain the properties for the specified hostGroupName.

**Description**  
The stmfGetHostGroupMembers() function retrieves the properties of the specified host group. The caller should call stmfFreeMemory(3STMF) when this list is no longer needed.

**Return Values**  
The following values are returned:

- **STMF_ERROR_NOT_FOUND**  
The specified hostGroupName was not found in the system.

- **STMF_STATUS_SUCCESS**  
The API call was successful.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
stmfFreeMemory(3STMF), libstmf(3LIB), attributes(5)
stmfGetLogicalUnitList(3STMF)

Name  stmfGetLogicalUnitList – retrieve the list of logical units

Synopsis  cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetLogicalUnitList(stmfGuidList **logicalUnitList);

Parameters  logicalUnitList  A pointer to a pointer to an stmfGuidList structure. On successful return, this will contain a list of logical units in the system.

Description  The stmfGetLogicalUnitList() function retrieves the list of logical units. The caller should call stmfFreeMemory(3STMF) when this list is no longer needed.

Return Values  The following values are returned:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMF_ERROR_NOMEM</td>
<td>The library was unable to allocate sufficient memory for logicalUnitList.</td>
</tr>
<tr>
<td>STMF_STATUS_SUCCESS</td>
<td>The API call was successful.</td>
</tr>
</tbody>
</table>

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  stmfFreeMemory(3STMF), libstmf(3LIB), attributes(5)
stmfGetLogicalUnitProperties – retrieve the properties of the specified logical unit

**Synopsis**
```
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetLogicalUnitProperties(stmfGuid *logicalUnit,
                                 stmfLogicalUnitProperties *logicalUnitProps);
```

**Parameters**
- `logicalUnit` The identifier of the logical unit whose properties are being retrieved.
- `logicalUnitProps` A pointer to an stmfLogicalUnitProperties structure. On successful return, this will contain the properties for the specified `logicalUnitOid`.

**Description**
The `stmfGetLogicalUnitProperties()` function retrieves the properties of the specified logical unit.

**Return Values**
The following values are returned:
- `STMF_ERROR_LOGICAL_UNIT_NOT_REGISTERED` The `logicalUnit` is not a valid registered logical unit in the system.
- `STMF_STATUS_SUCCESS` The API call was successful.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**
libstmf(3LIB), attributes(5)
stmfGetLuResource (3STMF)

Name
stmfGetLuResource – get a logical unit resource for a currently registered logical unit

Synopsis
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetLuResource(stmfGuid *luGuid, luResource *hdl);

Parameters
luGuid The guid of logical unit to retrieve.
hdI The logical unit resource to create.

Description
The stmfGetLuResource() function retrieves a logical unit resource hdl for a given logical
unit. The luGuid argument must represent a currently registered stmf logical unit. This
retrieved resource is a set of device-specific properties for a logical unit device. This allocates
an luResource hdl of device type matching luGuid. The stmfFreeLuResource(3STMF)
function should be used when hdl is no longer needed.

Return Values
The following values are returned:
STMF_STATUS_SUCCESS The API call was successful.
STMF_ERROR_NOT_FOUND The guid does not exist.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also
libstmf(3LIB), stmfFreeLuResource(3STMF), attributes(5)
### Name
stmfGetPersistMethod – get the current persistence method for stmfs

### Synopsis
```
c c [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetPersistMethod(uint8_t *persistType, boolean_t serviceState);
```

### Parameters
- **persistType**: On success, contains the current persistence setting based on `serviceState`.
- **serviceState**: When set to `B_TRUE`, `persistType` will contain the persist method currently set for the service. When set to `B_FALSE`, `persistType` will contain the persist method for the current library open.

### Description
The `stmfGetPersistMethod()` function retrieves the current persistent method setting for the service or for a given library open. When set to `B_TRUE`, retrieves the setting from the service.

### Return Values
The following values are returned:
- **STMF_STATUS_SUCCESS**: The API call was successful.
- **STMF_ERROR_PERSIST_TYPE**: Unable to retrieve persist type from service.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

### See Also
- `libstmf(3LIB)`, `attributes(5)`
stmfGetProviderData(3STMF)

**Name**
stmfGetProviderData – retrieve the data for the specified provider

**Synopsis**

c c [ flag... ] file... -lstmf [ library... ]

#include <libstmf.h>

int stmfGetProviderData(char *providerName, nvlist_t **nvl,
int providerType);

**Parameters**

- **providerName**
The name of the provider for which data is being retrieved.
- **nvl**
A pointer to a pointer to an nvlist_t. On success, this will contain the
nvlist retrieved. Caller is responsible for freeing the returned nvlist by
calling `nvlist_free(3NVPAIR)`.
- **providerType**
The value for this parameter must be either `STMF_LU_PROVIDER_TYPE` or
`STMF_PORT_PROVIDER_TYPE`.

**Description**
The `stmfGetProviderData()` function retrieves the data for the specified provider.

**Return Values**
The following values are returned:

- **STMF_ERROR_NOMEM**
The library was unable to allocate sufficient memory to return the
data.
- **STMF_STATUS_SUCCESS**
The API call was successful.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed (Obsolele)</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**
libstmf(3LIB), nvlist_free(3NVPAIR), stmfGetProviderDataProt(3STMF),
attributes(5)

**Notes**
The `stmfGetProviderData()` function is deprecated in favor of
`stmfGetProviderDataProt(3STMF)` and may be removed in a future revision of
libstmf(3LIB).
stmfGetProviderDataProt(3STMF)

Name  stmfGetProviderDataProt – retrieve data for the specified provider

Synopsis  cc { flag... } file... -lstmf [ library... ]
            #include <libstmf.h>

            int stmfGetProviderDataProt(char *providerName, nvlist_t **nvl,
                                        int providerType, uint64_t *token);

Parameters  

providerName  The name of the provider for which data is being retrieved.

nvl  A pointer to a pointer to an nvlist_t. On success, this will contain the
     nvlist retrieved. The caller is responsible for freeing the returned nvlist by
     calling nvlist_free(3NVPAIR).

providerType  The value for this parameter must be either STMF_LU_PROVIDER_TYPE or
               STMF_PORT_PROVIDER_TYPE.

token  A pointer to a uint64_t allocated by the caller. On success, this will contain
        a token for the returned data that can be used in a call to
        stmfSetProviderDataProt(3STMF) to ensure that the data returned in
        this call is not stale. If this value is NULL, the token will be ignored.

Description  The stmfGetProviderDataProt() function retrieves the data for the specified provider.

Return Values  The following values are returned:

STMF_ERROR_NOMEM  The library was unable to allocate sufficient memory to return the
                   data.

STMF_STATUS_SUCCESS  The API call was successful.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libstmf(3LIB), nvlist_free(3NVPAIR), stmfSetProviderDataProt(3STMF),
          attributes(5)
stmfGetState(3STMF)

Name      stmfGetState – retrieve the list of sessions on a target

Synopsis  cc [ flag... ] file... -lstmf [ library... ]
           #include <libstmf.h>

           int stmfGetState(stmfState *state);

Parameters state   A pointer to an stmfState structure allocated by the caller.

Description The stmfGetState() function retrieves the list of target port groups.

Return Values The following values are returned:
    STMF_STATUS_SUCCESS   The API call was successful.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libstmf(3LIB), attributes(5)
stmfGetStmfProp(3STMF)

Name  stmfGetStmfProp – retrieve default stmf properties for luns and targets

Synopsis  cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetStmfProp(uint8_t propType, char *propVal, size_t *propLen);

Parameters  propType  a property type value. See DESCRIPTION for valid values.
propVal  a property value
propLen  the length of the specified property value. If propLen was of an insufficient size
to hold the returned property value, propLen will contain the required size of
the buffer and STMF_ERROR_INVALID_ARG will be returned.

Description  This function gets the default properties for the specified property type. All property values
are expressed in human-readable form. The propType argument can be one of the following
values:

<table>
<thead>
<tr>
<th>STMF_DEFAULT_LU_STATE</th>
<th>Retrieve the current default state for luns. The default value is “online”.</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMF_DEFAULT_TARGET_PORT_STATE</td>
<td>Retrieve the current default state for target ports. The default value is “online”.</td>
</tr>
</tbody>
</table>

Return Values  The following values are returned:

<table>
<thead>
<tr>
<th>STMF_STATUS_SUCCESS</th>
<th>The API call was successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMF_ERROR_INVALID_ARG</td>
<td>Either the propType or propVal argument is invalid.</td>
</tr>
<tr>
<td>STMF_ERROR_NOT_FOUND</td>
<td>The specified propType was not found in the system.</td>
</tr>
</tbody>
</table>

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libstmf(3LIB), stmfSetStmfProp(3STMF), attributes(5)
stmfGetTargetGroupList() - retrieve the list of target port groups

Synopsis

cc [ flag... ] file... -lstmf [ library... ]

#include <libstmf.h>

int stmfGetTargetGroupList(stmfGroupList **targetGroupList);

Parameters
targetGroupList A pointer to a pointer to an stmfGroupList structure. On successful return, this will contain a list of target port group object identifiers.

Description

The stmfGetTargetGroupList() function retrieves the list of target port groups. The caller should call stmfFreeMemory(3STMF) when this list is no longer needed.

Return Values

The following values are returned:

STMF_ERROR_NOMEM The library was unable to allocate sufficient memory for targetGroupList.

STMF_STATUS_SUCCESS The API call was successful.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also

libstmf(3LIB), stmfFreeMemory(3STMF), attributes(5)
Name
stmfGetTargetGroupMembers – retrieve the properties of the specified target port group

Synopsis
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetTargetGroupMembers(stmfGroupName *targetGroupName,
    stmfGroupProperties **groupProperties);

Parameters
targetGroupName The name of the target port group whose member list is being retrieved.
groupProperties A pointer to a pointer to an stmfGroupProperties structure. On successful return, this will contain the properties for the specified targetGroupName.

Description
The stmfGetTargetGroupMembers() function retrieves the properties of the specified target port group. The caller should call stmfFreeMemory(3STMF) when this list is no longer needed.

Return Values
The following values are returned:

- STMF_ERROR_NOT_FOUND The specified targetGroupName was not found in the system.
- STMF_STATUS_SUCCESS The API call was successful.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also
libstmf(3LIB), stmfFreeMemory(3STMF), attributes(5)
Name  stmfGetTargetList – retrieve the list of target ports

Synopsis  cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetTargetList(stmfDevidList **targetList);

Parameters  targetList  A pointer to a pointer to an stmfDevidList structure. On successful return, this will contain a list of target ports in the system.

Description  The stmfGetTargetList() function retrieves the list of target ports. The caller should call stmfFreeMemory(3STMF) when this list is no longer needed.

Return Values  The following values are returned:

STMF_ERROR_NOMEM  The library was unable to allocate sufficient memory for targetList.
STMF_STATUS_SUCCESS  The API call was successful.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libstmf(3LIB), stmfFreeMemory(3STMF), attributes(5)
stmfGetTargetProperties(3STMF)

Name  stmfGetTargetProperties – retrieve the properties of the specified target port

Synopsis  cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetTargetProperties(stmfDevid *target,
                        stmfTargetProperties *targetProperties);

Parameters  target  The identifier of the target port whose properties are being retrieved.
            targetProperties  A pointer to an stmfTargetProperties structure allocated by the caller. On successful return, the structure will contain the properties for the specified.

Description  The stmfGetTargetProperties() function retrieves the properties of the specified target port.

Return Values  The following values are returned:
                   STMF_ERROR_NOT_FOUND  The specified target was not found in the system.
                   STMF_STATUS_SUCCESS  The API call was successful.

Attributes  See attributes(5) for descriptions of the following attributes:


<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libstmf(3LIB), attributes(5)
stmfGetViewEntryList(3STMF)

Name
stmfGetViewEntryList – retrieve the list of view entries for a specified logical unit

Synopsis
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetViewEntryList(stmfGuid *logicalUnit,
stmfViewEntryList **viewEntryList);

Parameters
logicalUnit The identifier of the logical unit for which to retrieve the list of view entries.
viewEntryList A pointer to a pointer to an stmfViewEntryList structure. On successful return, this will contain a list of view entries for logicalUnit.

Description
The stmfGetViewEntryList() function retrieves the list of view entries for a specified logical unit. The caller should call stmfFreeMemory(3STMF) when this list is no longer needed.

Return Values
The following values are returned:
STMF_ERROR_NOMEM The library was unable to allocate sufficient memory for viewEntryList.
STMF_STATUS_SUCCESS The API call was successful.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also
libstmf(3LIB), stmfFreeMemory(3STMF), attributes(5)
**Name**  
stmfGetViewLuNumberList – retrieve the list of logical-unit-numbers

**Synopsis**  
```c
#include <libstmf.h>

int stmfGetViewLuNumberList(viewResource resource, 
                          stmfLuNbrEntryList **luNbrList);
```

**Parameters**  
- **resource**  
  A view resource previously allocated using stmfGetViewResourceList(3STMF) call.

- **luNbrList**  
  A pointer to a pointer to structure stmfLuNbrEntryList. On successful return, this will contain information of list of mapped logical-unit-numbers and the corresponding initiator for the given logical unit.

**Description**  
The stmfGetViewLuNumberList() function retrieves the list of logical-unit-numbers mapped at that time for a specified logical unit for all initiators associated with the host group in a view entry. A view entry can be added to a logical unit either by requesting a system assigned automatic logical-unit-number or by specifying a fixed logical-unit-number. If a logical-unit-number was specified while adding the view entry, all the mapped logical-unit-numbers will be same for all initiators associated to the host group in the view entry throughout the existence of that view. If an assigned automatic logical-unit-number was requested, the system may remap the number for any initiator associated to the view to avoid any possible conflict, in which case logical-unit-numbers may vary across the initiators associated with the host group of the view.

The caller should call stmfFreeMemory(3STMF) when luNbrList data is no longer needed.

**Return Values**  
The following values are returned:

- **STMF_STATUS_SUCCESS**  
The API call was successful.

- **STMF_ERROR_INVALID_ARG**  
Invalid arguments passed

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
libstm(3LIB), stmfFreeMemory(3STMF), stmfGetViewResourceList(3STMF), attributes(5)
stmfGetViewProp(3STMF)

Name
stmfGetViewProp – get a view entry property of a logical unit

Synopsis
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfGetViewProp(viewResource resource, stmfViewPropType propType,
char *propVal, size_t *propLen);

Parameters
resource
A view resource previously allocated using stmfGetViewResourceList(3STMF) call.

propType
Property type value. The following values are supported:

STMF_VE_PROP_GUID
ASCII hexadecimal string of 32 characters representing the unique identifier for the
logical unit to which the specified view resource is attached.

STMF_VE_PROP_HG_ALL
Set to “true” if host group was not specified while adding the view so that the associated
logical unit is available to all initiators; otherwise it is set to “false”.

STMF_VE_PROP_HG
Host Group name attached to the specified view resource. This property is set to NULL if
STMF_VE_PROP_HG_ALL is set to “true”.

STMF_VE_PROP_TG_ALL
Set to “true” if target group was not specified while adding the view so that the
associated logical unit is available to all targets; otherwise it is set to “false”.

STMF_VE_PROP_TG
Target Group name attached to the specified view resource. This property is set to NULL if
STMF_VE_PROP_TG_ALL is set to “true”.

STMF_VE_PROP_IDX
View entry index. The numeric value of the view entry, assigned by the system per
logical unit.

STMF_VE_PROP_LUN_TYPE
Logical unit number type opted while adding the view entry, which is a directive for the
system to map the associated logical unit for all initiators belonging to the host group of
this view entry. The logical unit number type is either “fixed” or “auto” representing
either a fixed LUN number or system-assigned automatic logical unit number.

STMF_VE_PROP_LUN
Fixed logical unit number. This property is valid only if a fixed logical unit number was
opted while adding the view entry.

propVal
Pointer to property value. The memory for propVal needs to be allocated by the calling
function.
propLen
The length of the property value. With the `stmfGetViewProp()` call, this parameter needs to be set with the length of the buffer allocated for `propVal`. `propLen` will be set to actual length of `propVal` on function return. In the case of insufficient size to hold the requested property value, `propLen` will contain the actual size of the `propVal` and `stmfGetViewProp()` will return `STMF_ERROR_INVALID_ARG`.

**Description**
The `stmfGetViewProp()` function obtains the property values of the specified view resource.

**Return Values**
The following values are returned:
- `STMF_ERROR_INVALID_ARG`: One or more provided parameters are invalid.
- `STMF_ERROR_NO_PROP_VAL`: No value for the specified property
- `STMF_STATUS_SUCCESS`: The API call was successful.

**Attributes**
See [attributes(5)] for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**
`libstm(3LIB), stmfViewResourceList(3STMF), attributes(5)"
stmfGetViewResourceList(3STMF)

**Name**  
stmfGetViewResourceList – get a list of view entry resources for currently registered logical unit

**Synopsis**  
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

```c
int stmfGetViewResourceList(stmfGuid *logicalUnit,
viewResourceList **resourceList);
```

**Parameters**  
- **logicalUnit**  
The identifier of the logical unit to which the views entries are added.
- **resourceList**  
A pointer to a pointer to structure viewResourceList. On successful return, this will contain list of view entry resources.

**Description**  
The stmfGetViewResourceList() function retrieves the list of view resources representing view entries attached to a specified logical unit. The logicalUnit parameter must represent a currently registered STMF logical unit.

The caller should call stmfFreeViewResourceList(3STMF) when resourceList data is no longer needed.

**Return Values**  
The following values are returned:
- **STMF_STATUS_SUCCESS**  
The API call was successful.
- **STMF_ERROR_NOT_FOUND**  
The specified logical unit does not exist.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
libstm(3LIB), stmfFreeViewResourceList(3STMF), attributes(5)
Name  stmfImportLu - import a logical unit

Synopsis  cc { flag... } file... -lstmf [ library... ]
          #include <libstmf.h>

          int stmfImportLu(uint16_t dType, char *fname, stmfGuid *luGuid);

Parameters  dType   the device type of the logical unit being imported. Only STMF_DISK is currently supported.

            fname  the filename of the logical unit being imported

            luGuid pointer to a stmfGuid allocated by the caller. On success, this contains the guid of the imported logical unit. If luGuid is NULL, this parameter is ignored.

Description  The stmfImportLu() function imports a previously created logical unit. The fname argument must be set to the filename where the metadata for the logical unit is stored. See stmfCreateLu(3STMF).

Return Values  The following values are returned:

            STMF_STATUS_SUCCESS  The API call was successful.

            STMF_ERROR_INVALID_ARG  The dType or fname argument was invalid.

            STMF_ERROR_META_FILE_NAME  The specified meta file could not be accessed.

            STMF_ERROR_DATA_FILE_NAME  The data file could not be accessed.

Attributes  See attributes(5) for descriptions of the following attributes:


<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  stmfCreateLu(3STMF), attributes(5)
stmfInitProxyDoor(3STMF)

**Name**
stmfInitProxyDoor – establish the door server with the STMF proxy service

**Synopsis**
```c
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfInitProxyDoor(int *hdl, int fd);
```

**Parameters**
- `hdl` a pointer to an `int` that will contain the handle for the proxy door to be used in calls to `stmfPostProxyMsg(3STMF)` and `stmfDestroyProxyDoor(3STMF)`.
- `fd` the door file descriptor for the established door server

**Description**
The `stmfInitProxyDoor()` function establishes the door server with the STMF proxy service. The STMF proxy service is responsible for sending SCSI commands to the peer node on behalf of a logical unit in the Standby asymmetric logical unit access (ALUA) state. `stmfInitProxyDoor()` should be called once a peer-to-peer communication channel between the two participating ALUA nodes has been established by the caller.

The `door_call(3C)` from the STMF proxy service to the door server will fill in the `door_arg_t` structure as follows:
```c
door_arg_t arg;
uint32_t result;

arg.data_ptr = buf;
arg.data_size = size;
arg.desc_ptr = NULL;
arg.desc_num = 0;
arg.rbuf = (char *)&result
arg.rsize = sizeof (result);
```

The tuple `<data_ptr, data_size>` is expected to arrive at the peer node STMF proxy service via `stmfPostProxyMsg()`.

The door server is expected to complete the door call with these arguments to `door_return(3C)`:
```c
uint32_t result;

(void) door_return((char *)&result, sizeof(result), NULL, 0);
```

where `result` is of type `uint32_t` and set to `0` on success, non-zero on failure.

Non-zero values are logged as errors without further action. No file descriptors will be exchanged by the door call or return.

**Return Values**
The following values are returned:

- **STMF_ERROR_DOOR_INSTALLED** A previous door has already been established.
STMF_STATUS_SUCCESS The API call was successful.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also** door_call(3C), door_return(3C), libstmf(3LIB), stmfDestroyProxyDoor(3STMF), stmfPostProxyMsg(3STMF), attributes(5)
Name | stmfLuStandby – set the access state of a logical unit to standby mode  

Synopsis | cc [ flag... ] file... -lstmf [ library... ]  
#include <libstmf.h>  

int stmfSetAluaState(stmfGuid *luGuid);  

Parameters | luGuid | a pointer to an stmfGuid structure containing the guid of the logical unit to set to standby  

Description | The stmfLuStandby() function sets the access state of a logical unit to standby mode. When successfully set, a standby logical unit switches its asymmetric logical unit access state to a one of “Transition to Standby” (see stmfGetLuProp(3STMF)). Once moved to this state, the backing store for the logical unit will be released by the logical unit provider (std for disk devices). To move a logical unit out of “Standby” or the “Transition to Standby” state, stmfImportLu(3STMF) or the import-lu subcommand of stmfadm(1M) must be executed on the logical unit. On a successful logical unit import, the access state of the logical unit will move to “Active” in addition to sending a message to its peer that will complete the peer’s transition to “Standby”. The current access state for the logical unit can be retrieved using stmfGetLuProp() where the property type is STMF_LU_PROP_ACCESS_STATE.  

Return Values | The following values are returned:  
STMF_ERROR_NOT_FOUND | The guid does not exist.  
STMF_STATUS_SUCCESS | The API call was successful.  

Attributes | See attributes(5) for descriptions of the following attributes:  

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also | stmfadm(1M), libstmf(3LIB), stmfGetLuProp(3STMF), stmfImportLu(3STMF), attributes(5)
stmfModifyLu(3STMF)

Name
stmfModifyLu, stmfModifyLuByFname – modify a logical unit

Synopsis
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfModifyLu(stmfGuid *luGuid, uint32_t prop, const char *propVal)
int stmfModifyLu(uint16_t dType, const char *fname, uint32_t prop,
                 const char *propVal)

Parameters
luGuid  The guid of logical unit to modify.
fname  The filename of logical unit to modify.
dType  Type of logical unit. See stmfCreateLuResource(3STMF).
prop  A property type value. See DESCRIPTION for valid values.
propVal  A property value.

Description
The stmfModifyLu() and stmfModifyLuByFname() functions modify the properties of a logical unit device.

Valid properties for modify STMF_DISK:

STMF_LU_PROP_ACCESS_STATE
  Asymmetric access state for the logical unit. Set to one of:
  0  Active
  1  Transition to Active
  2  Standby
  3  Transition to Standby

STMF_LU_PROP_ALIAS
  Up to 255 characters representing a user defined name for the device.
  Default: Set to file name of backing store.

STMF_LU_PROP_SIZE
  Numeric value with optional suffix (for example, 100G, 1T) to specify unit of size.
  Default: Size of device specified in the STMF_LU_PROP_DATA_FILENAME property value.

STMF_LU_PROP_WRITE_CACHE_DISABLE
  Write back cache disable. When specified as "true" or "false", specifies write back cache disable behavior.
  Default: Writeback cache setting of the backing store device specified by STMF_LU_PROP_DATA_FILENAME.
STMF_LU_PROP_WRITE_PROTECT
Write protect bit. When specified as "true" or "false", specifies whether the device behaves as a write protected device.

Default: "false"

**Return Values**
The following values are returned:

- **STMF_STATUS_SUCCESS**: The API call was successful.
- **STMF_ERROR_INVALID_ARG**: Either prop or propVal is unrecognized.
- **STMF_ERROR_INVALID_PROPSIZE**: The size of propVal is invalid.
- **STMF_ERROR_INVALID_PROP**: The value of prop is unknown for this resource type.

**Attributes**
See [attributes(5)] for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**
libstmf(3LIB), stmfCreateLuResource(3STMF), attributes(5)
Name  stmfOfflineLogicalUnit – take offline a logical unit that is currently in the online state

Synopsis  
```
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfOfflineLogicalUnit(stmfGuid *logicalUnit);
```

Parameters  
- **logicalUnit**  The identifier of the logical unit to offline.

Description  
The `stmfOfflineLogicalUnit()` function takes offline a logical unit that is currently in the online state. Once in the offline state, the logical unit will no longer be capable of servicing requests in the system.

This API call can be used to take offline a logical unit for servicing. Once the logical unit is offline, an initiator port that attempts to issue any SCSI commands to the offlined logical unit will receive a check condition. For purposes of the REPORT LUNS command, the logical unit will no longer appear in the logical unit inventory for any initiator ports to which it is currently mapped by one or more view entries.

Return Values  
The following values are returned:

- **STMF_ERROR_BUSY**  The device is currently busy.
- **STMF_STATUS_SUCCESS**  The API call was successful.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  
libstmf(3LIB), attributes(5)
stmfOfflineTarget – take offline a target port that is currently in the online state

Synopsis

cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfOfflineTarget(stmfDevid *target);

Parameters
target  The identifier of the target port to offline.

Description

The stmfOfflineTarget() function takes offline a target port that is currently in the online state. Once in the offline state, the target port will no longer be capable of servicing requests in the system.

This API call can be used to take offline a target port device for servicing. Once the target port is offline, it will no longer be available to any entities outside of the SCSI Target Mode Framework. Any initiator ports that currently have sessions established by the offline target port will be logged out.

Return Values

The following values are returned:

- STMF_ERROR_BUSY  The device is currently busy.
- STMF_STATUS_SUCCESS  The API call was successful.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also

libstmf(3LIB), attributes(5)
The identifier of the logical unit to take online.

Description
The `stmfOnlineLogicalUnit()` function takes online of a logical unit that is currently in the offline state.

Return Values
The following values are returned:

- `STMF_STATUS_SUCCESS` The API call was successful.

Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also
libstmf(3LIB), attributes(5)
stmfOnlineTarget (3STMF)

Name  stmfOnlineTarget – take online a target port that is currently in the offline state

Synopsis  cc [ flag ... ] file... -lstmf [ library ... ]
#include <libstmf.h>

int stmfOnlineTarget(stmfDevid *target);

Parameters  target  The identifier of the target port to online.

Description  The stmfOnlineTarget() function takes online a target port that is currently in the offline state.

Return Values  The following values are returned:

STMF_STATUS_SUCCESS  The API call was successful.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  libstmf(3LIB), attributes(5)
The `stmfPostProxyMsg()` function passes down to the STMF proxy service the message received from the peer node's STMF proxy service door upcall.

The following values are returned:

- **STMF_ERROR_INVALID_ARG** The `buf` argument is NULL.
- **STMF_POST_MSG_FAILED** The attempt to post the message failed.
- **STMF_STATUS_SUCCESS** The API call was successful.

See also `stmfInitProxyDoor(3STMF), libstmf(3LIB), attributes(5)`
**stmfRemoveFromHostGroup**

**Name**  
stmfRemoveFromHostGroup – remove an initiator port from an host group

**Synopsis**  
```c
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfRemoveFromHostGroup(stmfGroupName *hostGroupName
                      stmfDevid *initiatorPortName);
```

**Parameters**  
- **hostGroupName**  
The name of the host group from which the specified `hostGroupName` is being removed.
- **initiatorPortName**  
The device identifier of the initiator port to remove from the specified host group.

**Description**  
The `stmfRemoveFromHostGroup()` function removes an initiator port from an host group.

**Return Values**  
The following values are returned:
- **STMF_ERROR_GROUP_NOT_FOUND**  
The specified `hostGroupName` was not found in the system.
- **STMF_ERROR_MEMBER_NOT_FOUND**  
The specified `initiatorPortName` was not found in the system.
- **STMF_STATUS_SUCCESS**  
The API call was successful.

**Attributes**  
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`libstmf(3LIB), attributes(5)`
Name  
stmfRemoveFromTargetGroup – remove a target port from an target port group

Synopsis  
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfRemoveFromTargetGroup(stmfGroupName *targetGroupName
stmfDevid *targetName);

Parameters  
targetGroupName  The name of the target port group from which the specified
targetGroupName is being removed.
targetName  The device identifier of the target port to remove from the specified
target port group.

Description  
The stmfRemoveFromTargetGroup() function removes a target port from an target port

Return Values  
The following values are returned:

STMF_ERROR_GROUP_NOT_FOUND  The specified targetGroupName was not found in the
system.
STMF_ERROR_MEMBER_NOT_FOUND  The specified targetName was not found in the system.
STMF_ERROR_TG_ONLINE  The specified targetName must be offline.
STMF_STATUS_SUCCESS  The API call was successful.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  
libstmf(3LIB), attributes(5)
**stmfRemoveViewEntry(3STMF)**

**Name**  
stmfRemoveViewEntry – remove a view entry from the system

**Synopsis**  
cc [ flag... ] file... -lstmf [ library... ]  
#include <libstmf.h>

```c
int stmfRemoveViewEntry(stmfGuid *logicalUnit,
                        uint32_t viewEntry);
```

**Parameters**  
- `logicalUnit`  
The identifier of the logical unit for the view entry being removed.
- `viewEntry`  
The numeric value of the view entry to be removed.

**Description**  
The `stmfRemoveViewEntry()` function removes a view entry from the system.

**Return Values**  
The following values are returned:
- `STMF_ERROR_NOT_FOUND`  
The specified `logicalUnit` or `viewEntryName` was not found in the system.
- `STMF_STATUS_SUCCESS`  
The API call was successful.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  
libstmf(3LIB), attributes(5)
Name
stmfSetAluaState – set the Asymmetric Logical Unit Access State (ALUA) mode for STMF

Synopsis
```
c { flag... } file... -lstmf [ library... ]
#include <libstmf.h>
int stmfSetAluaState(boolean_t alua_enabled, uint32_t node);
```

Parameters
- **alua_enabled**: B_TRUE when enabling ALUA mode; B_FALSE when disabling ALUA mode.
- **node**: Must be the value 0 or 1.

Description
The `stmfSetAluaState()` function sets the Asymmetric Logical Unit Access State (ALUA) mode for STMF. When `alua_enabled` is set to B_FALSE, `node` is ignored; otherwise, `node` must be set to 0 or 1. The `node` setting must be different for each node in a paired configuration. This function should be called only after the STMF proxy door service has been initialized (see `stmfInitProxyDoor(3STMF)`). When the ALUA state is enabled, all STMF logical units will be registered on the peer node as standby logical units. The standby logical units can then be exported to any SCSI initiator using the existing mechanisms in STMF, `stmfAddViewEntry(3STMF)` or the add-view subcommand of `stmfdm(1M)`. If ALUA mode is already enabled, it is valid to call this interface again with enabled set to B_TRUE. This action would result in re-initialization of the ALUA mode and can be used during recovery of a failed peer node.

Return Values
The following values are returned:
- `STMF_ERROR_INVALID_ARG`: Either `alua_enabled` or `node` was incorrectly set.
- `STMF_STATUS_SUCCESS`: The API call was successful.

Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also
stmfdm(1M), libstmf(3LIB), stmfAddViewEntry(3STMF), stmfInitProxyDoor(3STMF), attributes(5)
**Name**
stmfSetLuProp, stmfGetLuProp – set or get a logical unit property

**Synopsis**
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

```c
int stmfSetLuProp(luResource hdl, uint32_t prop, const char *propVal);
int stmfGetLuProp(luResource hdl, uint32_t prop, char *propVal, size_t *propLen);
```

**Parameters**
- **hdl** A logical unit resource previously allocated by stmfCreateLuResource(3STMF) or stmfGetLuResource(3STMF).
- **prop** A property type value. See DESCRIPTION for valid values.
- **propVal** A property value.
- **propLen** The length of the specified property value. If **propLen** was of an insufficient size to hold the returned property value, **propLen** will contain the required size of the buffer and STMF_ERROR_INVALID_ARG will be returned.

**Description**
These functions set or get property values. All property values are expressed in human-readable form. Boolean properties are expressed in case insensitive form of “true” or “false”. Properties that are represented by ASCII hexadecimal contain no leading characters to indicate a base hexadecimal representation (that is, no leading “0x”). The prop argument can be one of the following values:

**STMF_LU_PROP_ACCESS_STATE**
Asymmetric access state for the logical unit. Set to one of:
- 0 Active
- 1 Transition to Standby
- 2 Standby
- 3 Transition to Active

**STMF_LU_PROP_ALIAS**
Up to 255 characters representing a user defined name for the device.
Default: Set to file name of backing store.

**STMF_LU_PROP_BLOCK_SIZE**
Numeric value for block size in bytes in 2^n.
Default: 512

**STMF_LU_PROP_COMPANY_ID**
Organizational Unique Identifier. 6 hexadecimal ASCII characters representing the IEEE OUI company id assignment. This will be used to generate the device identifier (GUID).
Default: 00144F
STMF_LU_PROP_DATA_FILENAME
Character value representing the file name of the backing store device.

Default: None

STMF_LU_PROP_GUID
ASCII hexadecimal string of 32 characters representing the unique identifier for the device. This must be of valid 32 hexadecimal ASCII characters representing a valid NAA Registered Extended Identifier.

Default: Set by framework to a generated value.

STMF_LU_PROP_HOST_ID
8 hexadecimal ASCII characters representing the host ID assignment. This will be used to generate the globally unique identifier (GUID) for the logical unit.

Default: identifier returned by hostid(1).

STMF_LU_PROP_META_FILENAME
Metadata filename. When specified, will be used to hold the SCSI metadata for the logical unit.

Default: None. If this value is not specified, the value specified in STMF_LU_PROP_DATA_FILENAME will be used.

STMF_LU_PROP_MGMT_URL
Up to 1024 characters representing Management Network Address URLs. More than one URL can be passed using space delimited URLs.

Default: None

STMF_LU_PROP_PID
Up to 16 characters of product identification that will be reflected in the Standard INQUIRY data returned for the device.

Default: sSet to COMSTAR.

STMF_LU_PROP_SERIAL_NUM
Serial Number. Specifies the SCSI Vital Product Data Serial Number (page 80h). It is a character value up to 252 bytes in length.

Default: None

STMF_LU_PROP_SIZE
Numeric value w/optional suffix, e.g. 100G, 1T, to specify unit of size.

Default: Size of the device specified in the STMF_LU_PROP_DATA_FILENAME property value.

STMF_LU_PROP_VID
8 characters of vendor identification per SCSI SPC-3 and will be reflected in the Standard INQUIRY data returned for the device.
Defaul: Set to SUN.

STMF LU PROP WC_MODE_SELECT
Write cache setting changeable. Specified as “true” or “false”. When “true”, a SCSI MODE SELECT from the initiator can change the WRITE CACHE ENABLE bit on the caching mode page. When false, the WRITE CACHE ENABLE bit is not changeable. This setting does not impact the ability for stmfadm(1M) or libstmf(3LIB) to modify the write cache disable setting.

Default: “true”

STMF LU PROP WRITE_CACHE_DISABLE
Write back cache disable. When specified as “true” or “false”, specifies write back cache disable behavior.

Default: Writeback cache setting of the backing store device specified by STMF LU PROP_DATA_FILENAME.

STMF LU PROP WRITE_PROTECT
Write protect bit. When specified as “true” or “false”, specifies whether the device behaves as a write protected device.

Default: “false”

Return Values
The following values are returned:

STMF_STATUS_SUCCESS The API call was successful.
STMF_ERROR_INVALID_ARG Either prop or propVal is unrecognized.
STMF_ERROR_INVALID_PROPsize The size of propVal is invalid.
STMF_ERROR_INVALID_PROP The value of prop is unknown for this resource type.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also
hostid(1), stmfadm(1M), libstmf(3LIB), stmfCreateLuResource(3STMF), stmfGetLuResource(3STMF), attributes(5)
stmfSetPersistMethod - set persistence method for the stm service

Synopsis

```
cc [ flag... ] file... -lstmf [ library... ]
#include <libstmf.h>

int stmfSetPersistMethod(uint8_t persistType, boolean_t serviceSet);
```

Parameters

- **persistType**: The requested persistence setting. Can be either STMF_PERSIST_SMF or STMF_PERSIST_NONE.
- **serviceSet**: Set to indicate whether the setting should persist on the stm service. When set to B_FALSE, this setting is only applicable for the duration of the current library open or until a subsequent call is made to change the setting.

Description

The `stmfSetPersistMethod()` function sets the persistence method for stm.

Return Values

The following values are returned:

- **STMF_STATUS_SUCCESS**: The API call was successful.
- **STMF_ERROR_INVALID_ARG**: The `persistType` argument is invalid.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also

`libstmf(3LIB), attributes(5)`
stmfSetProviderData – set the data for the specified provider

**Synopsis**

```c
#include <libstmf.h>

int stmfSetProviderData(char *providerName, nvlist_t *nvl, int providerType);
```

**Parameters**

- `providerName`: The name of the provider for which data is being set.
- `nvl`: A pointer to an `nvlist_t` containing the nvlist to be set.
- `providerType`: The value must be either `STMF_LU_PROVIDER_TYPE` or `STMF_PORT_PROVIDER_TYPE`.

**Description**

The `stmfSetProviderData()` function sets the data for the specified provider.

**Return Values**

The following values are returned:

- `STMF_ERROR_NOMEM`: The library was unable to allocate sufficient memory to return the data.
- `STMF_STATUS_SUCCESS`: The API call was successful.

**Attributes**

See [attributes(5)](manpagessection3:ExtendedLibraryFunctions,Volume4 • LastRevised7Oct2008) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed (Obsolete)</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**

`libstmf(3LIB), stmfSetProviderDataProt(3STMF), attributes(5)`

**Notes**

The `stmfSetProviderData()` function is deprecated in favor of `stmfSetProviderDataProt(3STMF)` and may be removed in a future revision of `libstmf(3LIB).`
Name

stmfSetProviderDataProt – retrieve data for the specified provider

Synopsis

cc [ flag... ] file... -lstmf [ library... ]

#include <libstmf.h>

int stmfSetProviderDataProt(char *providerName, nvlist_t **nvl,
                         int providerType, uint64_t *token);

Parameters

providerName The name of the provider for which data is being set.
nvl A pointer to a pointer to an nvlist_t containing the nvlist to be set.
providerType The value for this parameter must be either STMF_LU_PROVIDER_TYPE or
             STMF_PORT_PROVIDER_TYPE.
token A pointer to a uint64_t that contains the value returned from a successful
      call to stmfGetProviderDataProt(3STMF). If this argument is NULL, the
      token is ignored. Otherwise, the token will be verified against the current
      data. If the token represents stale data, the call fails.

On success, token will contain the new token for the data being set and can
be used in subsequent calls to stmfSetProviderData(3STMF). On failure
the contents are undefined.

Description

The stmfSetProviderDataProt() function sets the data for the specified provider.

Return Values

The following values are returned:

STMF_ERROR_PROV_DATA_STALE The token value represents stale data.
STMF_STATUS_SUCCESS The API call was successful.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also

libstmf(3LIB), nvlist_free(3NVPAIR), stmfGetProviderDataProt(3STMF),
stmfSetProviderData(3STMF), attributes(5)
stmfSetStmfProp – set default stmf properties for luns and targets

Synopsis

cc [ flag... ] file... -lstmf [ library... ]

#include <libstmf.h>

int stmfSetStmfProp(uint8_t propType, char *propVal);

Parameters

propType  a property type value. See DESCRIPTION for valid values.

propVal   a property value

Description

This function sets the default properties for the specified property type. All property values are expressed in human-readable form. The propType argument can be one of the following values:

- **STMF_DEFAULT_LU_STATE**: Set the default state for luns. The new setting will only take effect after a service enable/restart on the stmf service. Valid propVal values are “online” or “offline”.

- **STMF_DEFAULT_TARGET_PORT_STATE**: Sets the default state for target ports. The new setting will only take effect after a service enable/restart on the stmf service. Valid propVal values are “online” or “offline”.

Return Values

The following values are returned:

- **STMF_STATUS_SUCCESS**: The API call was successful.
- **STMF_ERROR_INVALID_ARG**: The propType argument is invalid.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also

libstm(3LIB), stmfGetStmfProp(3STMF), attributes(5)
stmfValidateView–remove a target port from an target port group

Synopsis

```c
cc [ flag... ] file... lsmf [ library... ]
#include <libstmf.h>

int stmfValidateView(stmfViewEntry *view);
```

Parameters

- `view`: The view entry to validate or get the logical number.

Description

The `stmfValidateView()` function validates the logical unit number. This is done by setting `view->luNbrValid` to `B_TRUE` and setting `view->luNbr` to the logical unit number. A valid logical unit number is in the range of 0-16383.

The `stmfValidateView()` function finds the next available logical unit number by setting `view->luNbrValid` to `B_FALSE`. On success, the available logical unit number is returned in `view->luNbr`. A logical unit number is considered to be available if it is not currently consumed by an existing view entry where the target group and host group matches the view entry passed into this function. Until the logical unit number is no longer available, any calls to this function will get the same logical unit number in `view->luNbr`.

Return Values

The following values are returned:

- `STMF_ERROR_LUN_IN_USE`: The specified logical unit number is already in use for this logical unit.
- `STMF_STATUS_SUCCESS`: The API call was successful.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also `libstmf(3LIB), stmfAddViewEntry(3STMF), attributes(5)`
stobsl() and stobclear() functions translate character-coded labels into binary labels. They also modify an existing binary label by incrementing or decrementing it to produce a new binary label relative to its existing value.

The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to perform label translation on character-coded labels that dominate the process's sensitivity label.

The generic form of an input character-coded label string is:

```
[ + ] classification name ] [ [ + | - ] word ... ]
```

Leading and trailing white space is ignored. Fields are separated by white space, a '/' (slash), or a ',' (comma). Case is irrelevant. If `string` starts with `+` or `-`, `string` is interpreted a modification to an existing label. If `string` starts with a classification name followed by `+` or `-`, the new classification is used and the rest of the old label is retained and modified as specified by `string`. `+` modifies an existing label by adding words. `-` modifies an existing label by removing words. To the maximum extent possible, errors in `string` are corrected in the resulting binary label `label`.

The `stobsl()` and `stobclear()` functions also translate hexadecimal label representations into binary labels (see `hextob(3TSOL)`) when the string starts with `0x` and either `NEW_LABEL` or `NO_CORRECTION` is specified in `flags`.

The `flags` argument can take the following values:

- **NEW_LABEL**: `label` contents is not used, is formatted as a label of the relevant type, and is assumed to be `ADMIN_LOW` for modification changes. If `NEW_LABEL` is not present, `label` is validated as a defined label of the correct type dominated by the process's sensitivity label.

- **NO_CORRECTION**: No corrections are made if there are errors in the character-coded label `string`. `string` must be complete and contain all the label components that are required by the `label_encodings` file. The `NO_CORRECTION` flag implies the `NEW_LABEL` flag.

- **0 (zero)**: The default action is taken.

The `error` argument is a return parameter that is set only if the function is unsuccessful.
The `stobs1()` function translates the character-coded sensitivity label string into a binary sensitivity label and places the result in the return parameter `label`.

The `flags` argument can be either `NEW_LABEL`, `NO_CORRECTION`, or 0 (zero). Unless `NO_CORRECTION` is specified, this translation forces the label to dominate the minimum classification, and initial compartments set that is specified in the `label_encodings` file and corrects the label to include other label components required by the `label_encodings` file, but not present in `string`.

The `stobclear()` function translates the character-coded clearance string into a binary clearance and places the result in the return parameter `clearance`.

The `flags` argument can be either `NEW_LABEL`, `NO_CORRECTION`, or 0 (zero). Unless `NO_CORRECTION` is specified, this translation forces the label to dominate the minimum classification, and initial compartments set that is specified in the `label_encodings` file and corrects the label to include other label components that are required by the `label_encodings` file, but not present in `string`. The translation of a clearance might not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file that might contain different words and constraints.

**Return Values**
These functions return 1 if the translation was successful and a valid binary label was returned. Otherwise they return 0 and the value of the `error` argument indicates the error.

**Errors**
When these functions return zero, `error` contains one of the following values:

- 1  Unable to access the `label_encodings` file.
- 0  The label `label` is not valid for this translation and the `NEW_LABEL` or `NO_CORRECTION` flag was not specified, or the label `label` is not dominated by the process’s sensitivity `label` and the process does not have `PRIV_SYS_TRANS_LABEL` in its set of effective privileges.
- >0 The character-coded label `string` is in error. `error` is a one-based index into `string` indicating where the translation error occurred.

**Files**
`/etc/security/tsol/label_encodings`

The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
The stobs1() and stobclear() functions are obsolete. Use the str_to_label(3TSOL) function instead.

See Also  blcompare(3TSOL), hextob(3TSOL), libtsol(3LIB), str_to_label(3TSOL), attributes(5)

Notes  The functionality described on this manual page is available only if the system is configured with Trusted Extensions.

In addition to the ADMIN_LOW name and ADMIN_HIGH name strings defined in the label_encodings file, the strings "ADMIN_LOW" and "ADMIN_HIGH" are always accepted as character-coded labels to be translated to the appropriate ADMIN_LOW and ADMIN_HIGH label, respectively.

Modifying an existing ADMIN_LOW label acts as the specification of a NEW_LABEL and forces the label to start at the minimum label that is specified in the label_encodings file.

Modifying an existing ADMIN_HIGH label is treated as an attempt to change a label that represents the highest defined classification and all the defined compartments that are specified in the label_encodings file.

The NO_CORRECTION flag is used when the character-coded label must be complete and accurate so that translation to and from the binary form results in an equivalent character-coded label.
str_to_label() function is a simple function to parse human readable strings into labels of the requested type.

The string argument is the string to parse. If string is the result of a label_to_str() conversion of type M_INTERNAL, flags are ignored, and any previously parsed label is replaced.

If *label is NULL, str_to_label() allocates resources for label and initializes the label to the label_type that was requested before parsing string.

If *label is not NULL, the label is a pointer to a mandatory label that is the result of a previously parsed label and label_type is ignored. The type that is used for parsing is derived from label for any type-sensitive operations.

If flags is L_MODIFY_EXISTING, the parsed string can be used to modify this label.

If flags is L_NO_CORRECTION, the previously parsed label is replaced and the parsing algorithm does not attempt to infer missing elements from string to compose a valid label.

If flags is L_DEFAULT, the previously parsed label is replaced and the parsing algorithm makes a best effort to imply a valid label from the elements of string.

If flags contains L_CHECK_AR logically OR-ed with another value, the resulting label will be checked to ensure that it is within the “Accreditation Range” of the DIA encodings schema. This flag is interpreted only for MAC_LABEL label types.

The caller is responsible for freeing the allocated resources by calling the m_label_free() function. label_type defines the type for a newly allocated label. The label type can be:

MAC_LABEL The string should be translated as a Mandatory Access Control (MAC) label.
USER_CLEAR The string should be translated as a label that represents the least upper bound of the labels that the user is allowed to access.

If error is NULL, do not return additional error information for EINVAL. The calling process must have mandatory read access to label and human readable string. Or the calling process must have the sys_trans_label privilege.

The manifest constants ADMIN_HIGH and ADMIN_LOW are the human readable strings that correspond to the Trusted Extensions policy admin_high and admin_low label values. See labels(5).
Upon successful completion, the `str_to_label()` function returns 0. Otherwise, -1 is returned, `errno` is set to indicate the error, and `error` provides additional information for EINVAL. Otherwise, `error` is a zero-based index to the string parse failure point.

The `str_to_label()` function will fail if:

- **EINVAL** Invalid parameter. M_BAD_STRING indicates that `string` could not be parsed. M_BAD_LABEL indicates that the label passed in was in error. M_OUTSIDE_AR indicates that the resulting label is not within the "Accreditation Range" specified in the DIA encodings schema.
- **ENOTSUP** The system does not support label translations.
- **ENOMEM** The physical limits of the system are exceeded by size bytes of memory which cannot be allocated.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Standard</td>
<td>See below.</td>
</tr>
</tbody>
</table>

Parsing types that are relative to Defense Intelligence Agency (DIA) encodings schema are Standard. Standard is specified in `label_encodings(4)`.

See Also `label_to_str(3TSOL), libtsol(3LIB), m_label(3TSOL), label_encodings(4), attributes(5), labels(5)`

A number of the parsing rules rely on the DIA label encodings schema. The rules might not be valid for other label schemata.

The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
### Name
sysevent_bind_handle, sysevent_unbind_handle – bind or unbind subscriber handle

### Synopsis

```c
#include <libsysevent.h>

sysevent_handle_t *sysevent_bind_handle(void (*event_handler)(sysevent_t *ev));

void sysevent_unbind_handle(sysevent_handle_t *sysevent_hdl);
```

### Parameters
- **ev** pointer to sysevent buffer handle
- **event_handler** pointer to an event handling function
- **sysevent_hdl** pointer to a sysevent subscriber handle

### Description
The `sysevent_bind_handle()` function allocates memory associated with a subscription handle and binds it to the caller’s `event_handler`. The `event_handler` is invoked during subsequent system event notifications once a subscription has been made with `sysevent_subscribe_event(3SYSEVENT)`.

The system event is represented by the argument `ev` and is passed as an argument to the invoked event delivery function, `event_handler`.

Additional threads are created to service communication between `syseventd(1M)` and the calling process and to run the event handler routine, `event_handler`.

The `sysevent_unbind_handle()` function deallocates memory and other resources associated with a subscription handle and deactivates all system event notifications for the calling process. All event notifications are guaranteed to stop upon return from `sysevent_unbind_handle()`.

### Return Values
The `sysevent_bind_handle()` function returns a valid sysevent subscriber handle if the handle is successfully allocated. Otherwise, NULL is returned and `errno` is set to indicate the error.

The `sysevent_unbind_handle()` function returns no value.

### Errors
The `sysevent_bind_handle()` function will fail if:
- **EACCES** The calling process has an ID other than the privileged user.
- **EBUSY** There are no resources available.
- **EINVAL** The pointer to the function `event_handler` is NULL.
- **EMFILE** The process has too many open descriptors.
- **ENOMEM** There are insufficient resources to allocate the handle.
Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also syseventd(1M), sysevent_subscribe_event(3SYSEVENT), attributes(5)

Notes

Event notifications are revoked by syseventd when the bound process dies. Event notification is suspended if a signal is caught and handled by the event_handler thread. Event notification is also suspended when the calling process attempts to use fork(2) or fork1(2). Event notifications might be lost during suspension periods.

The libsysevent interfaces do not work at all in non-global zones.
**Name** sysevent_free – free memory for sysevent handle

**Synopsis**

```c
cc [flag...] file... -lsysevent [library...]
#include <libsysevent.h>

void sysevent_free(sysevent_t *ev);
```

**Parameters**

- `ev` handle to event an event buffer

**Description**

The sysevent_free() function deallocates memory associated with an event buffer.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** attributes(5)

**Notes**

The libsysevent interfaces do not work at all in non-global zones.
The `sysevent_get_attr_list()` function updates `attr_list` to point to a searchable name-value pair list associated with the `sysevent` event, `ev`. The interface manages the allocation of the attribute list, but it is up to the caller to free the list when it is no longer needed with a call to `nvlist_free()`. See `nvlist_alloc(3NVPAIR)`.

The `sysevent_get_attr_list()` function returns 0 if the attribute list for `ev` is found to be valid. Otherwise it returns −1 and sets `errno` to indicate the error.

The `sysevent_get_attr_list()` function will fail if:

- **ENOMEM** Insufficient memory available to allocate an `nvlist`.
- **EINVAL** Invalid `sysevent` event attribute list.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `syseventd(1M), nvlist_alloc(3NVPAIR), nvlist_lookup_boolean(3NVPAIR), attributes(5)`

Notes The libsysevent interfaces do not work at all in non-global zones.
# sysevent_get_class_name

*Function*: `char *sysevent_get_class_name(sysevent_t *ev);`

*Description*: Returns the class name of the event.

*Parameters*: `ev`: handle to event

*Examples*:
```
/* Filter on class and subclass */
if (strcmp(EC_PRIV, sysevent_get_class_name(ev)) != 0) {
```

## Examples

### Example 1

Parse `sysevent` header information.

The following example parses `sysevent` header information from an application's event handler.

```c
hrtime_t last_ev_time;
uint64_t last_ev_seq;

void
event_handler(sysevent_t *ev)
{
    sysevent_t *new_ev;
    int ev_sz;
    hrtime_t ev_time;
    uint64_t ev_seq;

    /* Filter on class and subclass */
    if (strcmp(EC_PRIV, sysevent_get_class_name(ev)) != 0) {
```
EXAMPLE 1 Parse sysevent header information. (Continued)

    return;
}

/*
 * Check for replayed sysevent, time must
 * be greater than previously recorded.
 */

    sysevent_get_event_time(ev, &ev_time);
    ev_seq = sysevent_get_seq(ev);
    if (ev_time < last_ev_time \\
        || (ev_time == last_ev_time && ev_seq <= last_ev_seq)) {
        return;
    }

    last_ev_time = ev_time;
    last_ev_seq = ev_seq;

    /* Store event for later processing */
    ev_sz = sysevent_get_size(ev):
    new_ev (sysevent_t *)malloc(ev_sz);
    bcopy(ev, new_ev, ev_sz);
    queue_event(new_ev);
}

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  attributes(5)

Notes  The libsysevent interfaces do not work at all in non-global zones.
The `sysevent_get_vendor_name()` function returns the vendor string for the publishing application or kernel subsystem that generated the system event. This information is useful for filtering sysevents for one or more vendors.

The interface manages the allocation of the vendor and publisher name strings, but it is the caller’s responsibility to free the strings when they are no longer needed by calling `free(3MALLOC)`. If the new vendor and publisher name strings cannot be created, `sysevent_get_vendor_name()` and `sysevent_get_pub_name()` return a null pointer and may set `errno` to `ENOMEM` to indicate that the storage space available is insufficient.

Examples

**EXAMPLE 1** Parse sysevent header information.

The following example parses sysevent header information from an application’s event handler.

```c
char *vendor;
char *pub;

void
event_handler(sysevent_t *ev)
{
    if (strcmp(EC_PRIV, sysevent_get_class_name(ev)) != 0) {
        return;
    }

    vendor = sysevent_get_vendor_name(ev);
}```
EXAMPLE 1 Parse sysevent header information. (Continued)

    if (strcmp("SUNW", vendor) != 0) {
        free(vendor);
        return;
    }
    pub = sysevent_get_pub_name(ev);
    if (strcmp("test_daemon", pub) != 0) {
        free(vendor);
        free(pub);
        return;
    }
    (void) kill(sysevent_get_pid(ev), SIGUSR1);
    free(vendor);
    free(pub);
}

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also malloc(3MALLOC), attributes(5)

Notes The libsysevent interfaces do not work at all in non-global zones.
**Name** sysevent_post_event – post system event for applications

**Synopsis**
```
cc [ flag... ] file... -lsysevent -lnvpair [ library... ]
#include <libsysevent.h>
#include <libnvpair.h>

int sysevent_post_event(char *class, char *subclass, char *vendor,
    char *publisher, nvlist_t *attr_list, sysevent_id_t *eid);
```

**Parameters**
- `attr_list` pointer to an `nvlist_t`, listing the name-value attributes associated with the event, or NULL if there are no such attributes for this event
- `class` pointer to a string defining the event class
- `eid` pointer to a system unique identifier
- `publisher` pointer to a string defining the event's publisher name
- `subclass` pointer to a string defining the event subclass
- `vendor` pointer to a string defining the vendor

**Description**
The `sysevent_post_event()` function causes a system event of the specified class, subclass, vendor, and publisher to be generated on behalf of the caller and queued for delivery to the sysevent daemon `syseventd(1M)`. The vendor should be the company stock symbol (or similarly enduring identifier) of the event posting application. The publisher should be the name of the application generating the event.

For example, all events posted by Sun applications begin with the company's stock symbol, “SUNW”. The publisher is usually the name of the application generating the system event. A system event generated by `devfsadm(1M)` has a publisher string of `devfsadm`.

The publisher information is used by sysevent consumers to filter unwanted event publishers.

Upon successful queuing of the system event, a unique identifier is assigned to `eid`.

**Return Values**
The `sysevent_post_event()` function returns 0 if the system event has been queued successfully for delivery. Otherwise it returns −1 and sets `errno` to indicate the error.

**Errors**
The `sysevent_post_event()` function will fail if:
- `ENOMEM` Insufficient resources to queue the system event.
- `EIO` The `syseventd` daemon is not responding and events cannot be queued or delivered at this time.
- `EINVAL` Invalid argument.
- `EPERM` Permission denied.
A copy error occurred.

**Examples**

**EXAMPLE 1** Post a system event event with no attributes.

The following example posts a system event event with no attributes.

```c
if (sysevent_post_event(EC_PRIV, "ESC_MYSUBCLASS", "SUNW", argv[0],
                       NULL), &eid == -1) {
    fprintf(stderr, "error logging system event\n");
}
```

**EXAMPLE 2** Post a system event with two name-value pair attributes.

The following example posts a system event event with two name-value pair attributes, an integer value and a string.

```c
nvlist_t *attr_list;
uint32_t uint32_val = 0XFFFFFFFF;
char *string_val = "string value data";
if (nvlist_alloc(&attr_list, 0, 0) == 0) {
    err = nvlist_add_uint32(attr_list, "uint32 data", uint32_val);
    if (err == 0)
        err = nvlist_add_string(attr_list, "str data", string_val);
    if (err == 0)
        err = sysevent_post_event(EC_PRIV, "ESC_MYSUBCLASS", "SUNW", argv[0], attr_list, &eid);
    if (err != 0)
        fprintf(stderr, "error logging system event\n");
    nvlist_free(attr_list);
}
```

**Attributes** See [attributes(5)] for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `devfsadm(1M), syseventd(1M), nvlist_add_boolean(3NVPAIR), nvlist_alloc(3NVPAIR), attributes(5)`

**Notes** The `libsysevent` interfaces do not work at all in non-global zones.
The \texttt{sysevent_subscribe_event()} function registers the caller's interest in event notifications belonging to the class \texttt{event\_class} and the subclasses contained in \texttt{event\_subclass\_list}. The subscriber handle \texttt{sysevent\_hdl} is updated with the new subscription and the calling process receives event notifications from the event handler specified in \texttt{sysevent\_bind\_handle}.

System events matching \texttt{event\_class} and a subclass contained in \texttt{event\_subclass\_list} published after the caller returns from \texttt{sysevent\_subscribe\_event()} are guaranteed to be delivered to the calling process. Matching system events published and queued prior to a call to \texttt{sysevent\_subscribe\_event()} may be delivered to the process's event handler.

The \texttt{num\_subclasses} argument provides the number of subclass string elements in \texttt{event\_subclass\_list}.

A caller can use the event class \texttt{EC\_ALL} to subscribe to all event classes and subclasses. The event class \texttt{EC\_SUB\_ALL} can be used to subscribe to all subclasses within a given event class.

Subsequent calls to \texttt{sysevent\_subscribe\_event()} are allowed to add additional classes or subclasses. To remove an existing subscription, \texttt{sysevent\_unsubscribe\_event()} must be used to remove the subscription.

The \texttt{sysevent\_unsubscribe\_event()} function removes the subscription described by \texttt{event\_class} for \texttt{sysevent\_hdl}. Event notifications matching \texttt{event\_class} will not be delivered to the calling process upon return.

A caller can use the event class \texttt{EC\_ALL} to remove all subscriptions for \texttt{sysevent\_hdl}. 
The library manages all subscription resources.

Return Values  The `sysevent_subscribe_event()` function returns 0 if the subscription is successful. Otherwise, −1 is returned and `errno` is set to indicate the error.

The `sysevent_unsubscribe_event()` function returns no value.

Errors  The `sysevent_subscribe_event()` function will fail if:

- EACCES  The calling process has an ID other than the privileged user.
- EINVAL  The `sysevent_hdl` argument is an invalid sysevent handle.
- ENOMEM  There is insufficient memory available to allocate subscription resources.

Examples  **EXAMPLE 1**  Subscribing for environmental events

```c
#include <libsysevent.h>
#include <sys/nvpair.h>

static int32_t attr_int32;

#define CLASS1 "class1"
#define CLASS2 "class2"
#define SUBCLASS_1 "subclass_1"
#define SUBCLASS_2 "subclass_2"
#define SUBCLASS_3 "subclass_3"
#define MAX_SUBCLASS 3

static void event_handler(sysevent_t *ev)
{
    nvlist_t *nvlist;

    /* Special processing for events (CLASS1, SUBCLASS_1) and 
     * (CLASS2, SUBCLASS_3)
     */
    if ((strcmp(CLASS1, sysevent_get_class_name(ev)) == 0 &&
        strcmp(SUBCLASS_1, sysevent_get_subclass_name(ev)) == 0) ||
        (strcmp(CLASS2, sysevent_get_subclass_name(ev)) == 0) &&
        strcmp(SUBCLASS_3, sysevent_get_subclass(ev)) == 0) {
        if (sysevent_get_attr_list(ev, &nvlist) != 0)
            return;
        if (nvlist_lookup_int32(nvlist, "my_int32_attr", &attr_int32)
            != 0)
            return;

        /* Event Processing */
    }
```
EXAMPLE 1  Subscribing for environmental events  (Continued)

} else {
    /* Event Processing */
}

int
main(int argc, char **argv)
{
    sysevent_handle_t *shp;
    const char *subclass_list[MAX_SUBCLASS];

    /* Bind event handler and create subscriber handle */
    shp = sysevent_bind_handle(event_handler);
    if (shp == NULL)
        exit(1);

    /* Subscribe to all CLASS1 event notifications */
    subclass_list[0] = EC_SUB_ALL;
    if (sysevent_subscribe_event(shp, CLASS1, subclass_list, 1) != 0) {
        sysevent_unbind_handle(shp);
        exit(1);
    }

    /* Subscribe to CLASS2 events for subclasses: SUBCLASS_1,
     * SUBCLASS_2 and SUBCLASS_3
     */
    subclass_list[0] = SUBCLASS_1;
    subclass_list[1] = SUBCLASS_2;
    subclass_list[2] = SUBCLASS_3;
    if (sysevent_subscribe_event(shp, CLASS2, subclass_list, MAX_SUBCLASS) != 0) {
        sysevent_unbind_handle(shp);
        exit(1);
    }

    for (; ; ) {
        (void) pause();
    }

Attributes  See attributes(5) for descriptions of the following attributes:
The `libsysysevent` interfaces do not work at all in non-global zones.

### See Also

- `sysyseventd(1M)`, `sysysevent_bind_handle(3SYSEVENT)`,
- `sysysevent_get_attr_list(3SYSEVENT)`, `sysysevent_get_class_name(3SYSEVENT)`,
- `sysysevent_get_vendor_name(3SYSEVENT)`, `attributes(5)`

### Notes

The `libsysysevent` interfaces do not work at all in non-global zones.
**Name**  
tsol_getrhtype – get trusted network host type

**Synopsis**  
```bash
c [flag...] file... -lttsnet [library...]
#include <libtsnet.h>

tsol_host_type_t tsol_getrhtype(char *hostname);
```

**Description**  
The `tsol_getrhtype()` function queries the kernel-level network information to determine the host type that is associated with the specified `hostname`. The `hostname` can be a regular hostname, an IP address, or a network wildcard address.

**Return Values**  
The returned value will be one of the enumerated types that is defined in the `tsol_host_type_t` typedef. Currently these types are `UNLABELED` and `SUN_CIPSO`.

**Files**  
`/etc/security/tsol/tnrhdb`  
Trusted network remote-host database

**Attributes**  
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
<td></td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**  
`libtsnet(3LIB), attributes(5)`

“Obtaining the Remote Host Type” in *Trusted Extensions Developer’s Guide*

**Notes**  
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
### Name
- `uuid_clear`, `uuid_compare`, `uuid_copy`, `uuid_generate`, `uuid_generate_random`, `uuid_generate_time`, `uuid_is_null`, `uuid_parse`, `uuid_time`, `uuid_unparse` – universally unique identifier (UUID) operations

### Synopsis
```c
#include <uuid/uuid.h>

void uuid_clear(uuid_t uu);
int uuid_compare(uuid_t uu1, uuid_t uu2);
void uuid_copy(uuid_t dst, uuid_t src);
void uuid_generate(uuid_t out);
void uuid_generate_random(uuid_t out);
void uuid_generate_time(uuid_t out);
int uuid_is_null(uuid_t uu);
int uuid_parse(char *in, uuid_t uu);
time_t uuid_time(uuid_t uu, struct timeval *ret_tv);
void uuid_unparse(uuid_t uu, char *out);
```

### Description
- **uuid_clear()** function sets the value of the specified universally unique identifier (UUID) variable `uu` to the NULL value.

- **uuid_compare()** function compares the two specified UUID variables `uu1` and `uu2` to each other. It returns an integer less than, equal to, or greater than zero if `uu1` is found to be, respectively, lexicographically less than, equal, or greater than `uu2`.

- **uuid_copy()** function copies the UUID variable `src` to `dst`.

- **uuid_generate()** function creates a new UUID that is generated based on high-quality randomness from `/dev/urandom`, if available. If `/dev/urandom` is not available, `uuid_generate()` calls `uuid_generate_time()`. Because the use of this algorithm provides information about when and where the UUID was generated, it could cause privacy problems for some applications.

- **uuid_generate_random()** function produces a UUID with a random or pseudo-randomly generated time and Ethernet MAC address that corresponds to a DCE version 4 UUID.

- **uuid_generate_time()** function uses the current time and the local Ethernet MAC address (if available, otherwise a MAC address is fabricated) that corresponds to a DCE version 1 UUID. If the UUID is not guaranteed to be unique, the multicast bit is set (the high-order bit of octet number 10).

- **uuid_is_null()** function compares the value of the specified UUID variable `uu` to the NULL value. If the value is equal to the NULL UUID, 1 is returned. Otherwise 0 is returned.
The `uuid_parse()` function converts the UUID string specified by `in` to the internal `uuid_t` format. The input UUID is a string of the form `cefa7a9c-1dd2-11b2-8350-880020adbeef`. In `printf(3C)` format, the string is “%08x-%04x-%04x-%04x-%012x”, 36 bytes plus the trailing null character. If the input string is parsed successfully, 0 is returned and the UUID is stored in the location pointed to by `uu`. Otherwise -1 is returned.

The `uuid_time()` function extracts the time at which the specified UUID `uu` was created. Since the UUID creation time is encoded within the UUID, this function can reasonably be expected to extract the creation time only for UUIDs created with the `uuid_generate_time()` function. The time at which the UUID was created, in seconds since January 1, 1970 GMT (the epoch), is returned (see `time(2)`). The time at which the UUID was created, in seconds and microseconds since the epoch is also stored in the location pointed to by `ret_tv` (see `gettimeofday(3C)`).

The `uuid_unparse()` function converts the specified UUID `uu` from the internal binary format to a string of the length defined in the `uuid.h` macro, `UUID_PRINTABLE_STRING_LENGTH`, which includes the trailing null character. The resulting value is stored in the character string pointed to by `out`.

**Attributes**  See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also**  `inetd(1M), time(2), gettimeofday(3C), libuuid(3LIB), printf(3C), attributes(5)`. 
# Name
volmgt_acquire – reserve removable media device

## Synopsis
```
cc [ flag ... ] file ... -lvolmgt [ library ... ]
#include <sys/types.h>
#include <volmgt.h>

int volmgt_acquire(char *dev, char *id, int ovr, char **err, pid_t *pidp);
```

## Description
This function is obsolete. The management of removable media by the Volume Management feature, including vol’d, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL web site. See hal(5). The return value of this function is undefined.

The `volmgt_acquire()` routine reserves the removable media device specified as `dev`. `volmgt_acquire()` operates in two different modes, depending on whether or not volume management is running.

If volume management is running, `volmgt_acquire()` attempts to reserve the removable media device specified as `dev`. Specify `dev` as either a symbolic device name or a physical device pathname.

If volume management is not running, `volmgt_acquire()` requires callers to specify a physical device pathname for `dev`. Specifying `dev` as a symbolic device name is not acceptable. In this mode, `volmgt_acquire()` relies entirely on the major and minor numbers of the device to determine whether or not the device is reserved.

If `dev` is free, `volmgt_acquire()` updates the internal device reservation database with the caller’s process id (`pid`) and the specified id string.

If `dev` is reserved by another process, the reservation attempt fails and `volmgt_acquire()`:

- sets `errno` to EBUSY
- fills the caller’s id value in the array pointed to by `err`
- fills in the `pid` to which the pointer `pidp` points with the `pid` of the process which holds the reservation, if the supplied `pidp` is non-zero.

If the override `ovr` is non-zero, the call overrides the device reservation.

## Return Values
The return from this function is undefined.

## Errors
The `volmgt_acquire()` routine fails if one or more of the following are true:

- EINVAL One of the specified arguments is invalid or missing.
- EBUSY `dev` is already reserved by another process (and `ovr` was not set to a non-zero value)
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

See Also  free(3C), malloc(3C), volmgt_release(3VOLMGT), attributes(5), hal(5)

Notes  When returning a string through err, volmgt_acquire() allocates a memory area using malloc(3C). Use free(3C) to release the memory area when no longer needed.

The ovr argument is intended to allow callers to override the current device reservation. It is assumed that the calling application has determined that the current reservation can safely be cleared.
This function is obsolete. The management of removable media by the Volume Management feature, including vold, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL web site. See hal(5). The return value of this function is undefined.

This routine asks volume Management to check the specified pathname and determine if new media has been inserted in that drive.

If a null pointer is passed in, then Volume Management will check each device it is managing that can be checked.

If new media is found, volmgt_check() tells volume management to initiate appropriate actions.

The return from this function is undefined.

This routine can fail, returning 0, if a stat(2) or open(2) of the supplied pathname fails, or if any of the following is true:

- ENXIO volume management is not running.
- EINTR An interrupt signal was detected while checking for media.

To check if any drive managed by volume management has any new media inserted in it:

```c
if (volmgt_check(NULL)) {
    (void) printf("Volume management found media\n");
}
```

This would also request volume management to take whatever action was appropriate for the new media.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
Since `volmgt_check()` returns 0 for two different cases (both when no media is found, and when an error occurs), it is up to the user to check `errno` to differentiate the two, and to ensure that volume management is running.

**See Also** `volcheck(1), open(2), stat(2), volmgt_inuse(3VOLMGT), volmgt_running(3VOLMGT), attributes(5), hal(5)`

**Notes** Since `volmgt_check()` returns 0 for two different cases (both when no media is found, and when an error occurs), it is up to the user to check `errno` to differentiate the two, and to ensure that volume management is running.
volmgt_feature_enabled(3VOLMGT)

Name  volmgt_feature_enabled – check whether specific Volume Management features are enabled

Synopsis  
cc [ flag ... ] file ... -l volmgt [ library ... ]  
#include <volmgt.h>

int volmgt_feature_enabled(char *feat_str);

Description  This function is obsolete. The management of removable media by the Volume Management feature, including vold, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL website. See hal(5). The return value of this function is undefined.

The volmgt_feature_enabled() routine checks whether specific volume management features are enabled. volmgt_feature_enabled() checks for the volume management features passed in to it by the feat_str parameter.

Return Values  The return from this function is undefined.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

See Also  volmgt_acquire(3VOLMGT), volmgt_release(3VOLMGT), attributes(5), hal(5)
volmgt_inuse() checks whether volume management is managing the specified pathname.

The return value of this function is undefined.

This routine can fail, returning 0, if a stat(2) of the supplied pathname or an open(2) of /dev/volctl fails, or if any of the following is true:

- ENXIO Volume management is not running.
- EINTR An interrupt signal was detected while checking for the supplied pathname for use.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

This routine requires volume management to be running.

Since volmgt_inuse() returns 0 for two different cases (both when a volume is not in use, and when an error occurs), it is up to the user to check errno to differentiate the two, and to ensure that volume management is running.
volmgt_ownspath(3VOLMGT)

Name
volmgt_ownspath – check volume management name space for path

Synopsis
cc [flag]... file... -lvolmgt [library]...
#include <volmgrt.h>

int volmgt_ownspath(char *path);

Parameters
path – A string containing the path.

Description
This function is obsolete. The management of removable media by the Volume Management feature, including void, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL web site. See hal(5). The return value of this function is undefined.

The volmgt_ownspath() function checks to see if a given path is contained in the volume management name space. This is achieved by comparing the beginning of the supplied path name with the output from volmgt_root(3VOLMGT)

Return Values
The return from this function is undefined.

Examples
EXAMPLE 1 Using volmgt_ownspath()

The following example first checks if volume management is running, then checks the volume management name space for path, and then returns the id for the piece of media.

char *path;

...

if (volmgt_running()) {
if (volmgt_ownspath(path)) {
(void) printf("id of %s is %lld\n", path, media_getid(path));
}
}

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

See Also
volmgt_root(3VOLMGT), volmgt_running(3VOLMGT), attributes(5), hal(5)
volmgt_release(3VOLMGT)

Name    volmgt_release – release removable media device reservation

Synopsis cc [ flag ... ] file ... -lvolmgt [ library ... ]
#include <volmgt.h>

int volmgt_release(char *dev);

Description This function is obsolete. The management of removable media by the Volume Management feature, including voltd, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL web site. See hal(5). The return value of this function is undefined.

The volmgt_release() routine releases the removable media device reservation specified as dev. See volmgt_acquire(3VOLMGT) for a description of dev.

If dev is reserved by the caller, volmgt_release() updates the internal device reservation database to indicate that the device is no longer reserved. If the requested device is reserved by another process, the release attempt fails and errno is set to 0.

Return Values The return from this function is undefined.

Errors On failure, volmgt_release() returns 0, and sets errno for one of the following conditions:

EINVAL dev was invalid or missing.
EBUSY dev was not reserved by the caller.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

See Also volmgt_acquire(3VOLMGT), attributes(5), hal(5)
volmgt_root – return the volume management root directory

Synopsis  cc [ flag... ] file... -lvolmgt [ library... ]

#include <volmgt.h>
const char *volmgt_root(void);

Description  This function is obsolete. The management of removable media by the Volume Management feature, including voltd, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL web site. See hal(5). The return value of this function is undefined.

The volmgt_root() function returns the current volume management root directory, which by default is /vol but can be configured to be in a different location.

Return Values  The return from this function is undefined.

Errors  This function may fail if an open() of /dev/volctl fails. If this occurs a pointer to the default Volume Management root directory is returned.

Examples  EXAMPLE 1  Finding the Volume Management Root directory.

To find out where the volume management root directory is:

if ((path = volmgt_root()) != NULL) {
    (void) printf("Volume Management root dir=%s\n", path);
} else {
    (void) printf("can't find Volume Management root dir\n");
}

Files  /dev  default location for the volume management root directory

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

See Also  open(2), volmgt_check(3VOLMGT), volmgt_inuse(3VOLMGT), volmgt_running(3VOLMGT), attributes(5), hal(5)

Notes  This function returns the default root directory location even when volume management is not running.
volmgt_running(3VOLMGT)

Name
volmgt_running – return whether or not volume management is running

Synopsis
cc [ flag... ] file... -lvolmgt [ library... ]
#include <volmgt.h>

int volmgt_running(void);

Description
This function is obsolete. The management of removable media by the Volume Management feature, including vold, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL web site. See hal(5).

volmgt_running() tells whether or not Volume Management is running.

Return Values
volmgt_running() always returns 0 indicating Volume Management (as implemented by vold) is not running.

Errors
volmgt_running() will fail, returning 0, if a stat(2) or open(2) of /dev/volctl fails, or if any of the following is true:

ENXIO  Volume Management is not running.
EINTR  An interrupt signal was detected while checking to see if Volume Management was running.

Examples
EXAMPLE 1 Using volmgt_running()
To see if Volume Management is running:

if (volmgt_running() != 0) {
  (void) printf("Volume Management is running\n");
} else {
  (void) printf("Volume Management is NOT running\n");
}

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

See Also
open(2), stat(2), volmgt_check(3VOLMGT), volmgt_inuse(3VOLMGT), attributes(5), hal(5)

Notes
Volume Management must be running for many of the Volume Management library routines to work.
volmgt_symname(3VOLMGT)

Name
volmgt_symname, volmgt_symdev – convert between Volume Management symbolic names, and the devices that correspond to them

Synopsis
cc [ flag... ] file... -lvolmgt [ library... ]
#include <volmgt.h>

char *volmgt_symname(char *pathname);
char *volmgt_symdev(char *symname);

Description
This function is obsolete. The management of removable media by the Volume Management feature, including vold, has been replaced by software that supports the Hardware Abstraction Layer (HAL). Programmatic support for HAL is through the HAL APIs, which are documented on the HAL web site. See hal(5). The return value of this function is undefined.

These two routines compliment each other, translating between Volume Management’s symbolic name for a device, called a symname, and the /dev pathname for that same device.

volmgt_symname() converts a supplied /dev pathname to a symname, Volume Management’s idea of that device’s symbolic name.

volmgt_symdev() does the opposite conversion, converting between a symname, Volume Management’s idea of a device’s symbolic name for a volume, to the /dev pathname for that device.

Return Values
The return from this function is undefined.

Errors
volmgt_symname() can fail, returning a null string pointer, if a stat(2) of the supplied pathname fails, or if an open(2) of /dev/volctl fails, or if any of the following is true:

ENXIO Volume Management is not running.
EINTR An interrupt signal was detected while trying to convert the supplied pathname to a symname.

volmgt_symdev() can fail if an open(2) of /dev/volctl fails, or if any of the following is true:

ENXIO Volume Management is not running.
EINTR An interrupt signal was detected while trying to convert the supplied symname to a /dev pathname.

Examples
EXAMPLE 1 Finding The symbolic name
This code finds out what symbolic name (if any) Volume Management has for /dev/rdsk/c0t6d0s2:

if ((nm = volmgt_symname("/dev/rdsk/c0t6d0s2"); == NULL) {
    (void) printf("path not managed\n");
} else {
    (void) printf("path managed as %s\n", nm);
}
EXAMPLE 1  Finding The symbolic name  

(Continued)

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

See Also  open(2), stat(2), free(3C), malloc(3C), volmgft_check(3VOLMGT), volmgft_inuse(3VOLMGT), volmgft_running(3VOLMGT), attributes(5), hal(5)
XTSOLgetClientAttributes – get all label attributes associated with a client

Synopsis

```
cc [flag...] file... -lX11 -lXtsol [library...]
#include <X11/extensions/Xtsol.h>

Status XTSOLgetClientAttributes(display, windowid, clientattr);

Display *display;
XID windowid;
XTsolClientAttributes *clientattrp;
```

Parameters

- `display`: Specifies a pointer to the `Display` structure. Is returned from `XOpenDisplay()`.
- `windowid`: Specifies window ID of X client.
- `clientattrp`: Client must provide a pointer to an `XTsolClientAttributes` structure.

Description

The `XTSOLgetClientAttributes()` function retrieves all label attributes that are associated with a client in a single call. The attributes include process ID, user ID, IP address, audit flags and session ID.

Return Values

None.

Errors

- BadAccess: Lack of privilege.
- BadValue: Not a valid client.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also

- `libXtsol(3LIB)`, `XTSOLgetPropAttributes(3XTSOL)`, `XTSOLgetResAttributes(3XTSOL)`, `attributes(5)`

Notes

The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
XTSOLgetPropAttributes – get the label attributes associated with a property hanging on a window

Synopsis
cc [flag... ] file... -lx11 -lxtsol [library... ]
#include <X11/extensions/Xtsol.h>

Status XTSOLgetPropAttributes(display, window, property, propattrp);

Display *display;
Window window;
Atom property;
XTSOLPropAttributes *propattrp;

Parameters
display Specifies a pointer to the Display structure; returned from XOpenDisplay().
window Specifies the ID of a window system object.
property Specifies the property atom.
propattrp Client must provide a pointer to XTSOLPropAttributes.

Description
The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. The XTSOLgetPropAttributes() function retrieves the label attributes that are associated with a property hanging out of a window in a single call. The attributes include UID and sensitivity label.

Return Values
None

Errors
BadAccess Lack of privilege
BadWindow Not a valid window
BadAtom Not a valid atom

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Unsafe</td>
</tr>
</tbody>
</table>

See Also
libxtsol(3LIB), XTSOLgetClientAttributes(3XTSOL),
XTSOLgetResAttributes(3XTSOL), attributes(5)

“Setting Window Polyinstantiation Information” in Trusted Extensions Developer’s Guide

Notes
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
 XT SOLgetPropLabel(3XT SOL)

Name    XT SOLgetPropLabel – get the label associated with a property hanging on a window

Synopsis cc [flag... ] file... -lX11 -lXT SOL [library... ]
#include <X11/extensions/Xtsol.h>

Status XT SOLgetPropLabel(display, window, property, sl);

Display *display;
Window window;
Atom property;
m_label_t *sl;

Parameters
display Specifies a pointer to the Display structure; returned from XOpenDisplay () .
window Specifies the ID of the window whose property's label you want to get.
property Specifies the property atom.
sl Returns a sensitivity label that is the current label of the specified property.

Description Client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. The XT SOLgetPropLabel () function retrieves the sensitivity label that is associated with a property hanging on a window.

Return Values None.

Errors BadAccess Lack of privilege.
BadWindow Not a valid window.
BadAtom Not a valid atom.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also libXT SOL(3LIB), XT SOLgetPropAttributes(3XT SOL), XT SOLsetPropLabel(3XT SOL), attributes(5)

“Setting Window Polyinstantiation Information” in Trusted Extensions Developer's Guide

Notes The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
**Name**  XTSOLgetPropUID – get the UID associated with a property hanging on a window

**Synopsis**  
cc [flag...] file... -lX11 -lXtsol [library...]  
#include <X11/extensions/Xtsol.h>  

Status XTSOLgetPropUID (display, window, property, uidp);

Display *display;  
Window window;  
Atom property;  
uid_t *uidp;

**Parameters**  
display  Specifies a pointer to the Display structure; returned from XOpenDisplay().

window  Specifies the ID of the window whose property’s UID you want to get.

property  Specifies the property atom.

uidp  Returns a UID which is the current UID of the specified property. Client needs to provide a uid_t type storage and passes the address of this storage as the function argument. Client must provide a pointer to uid_t.

**Description**  
The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. The XTSOLgetPropUID() function retrieves the ownership of a window’s property. This allows a client to get the ownership of an object it did not create.

**Return Values**  
None.

**Errors**  
BadAccess  Lack of privilege.

BadWindow  Not a valid window.

BadAtom  Not a valid atom.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**See Also**  
libXtsol(3LIB), XTSOLgetPropAttributes(3XTSOL), XTSOLsetPropUID(3XTSOL), attributes(5)

“Setting Window Polyinstantiation Information” in Trusted Extensions Developer’s Guide

**Notes**  
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
XTSOLgetResAttributes(3XTSOL)

Name
XTSOLgetResAttributes – get all label attributes associated with a window or a pixmap

Synopsis
cc [flag...] -I X11 -I xtsol [library...]
#include <X11/extensions/xtsol.h>

Status XTSOLgetResAttributes(display, object, type, winattrp);

Display *display;
XID object;
ResourceType type;
XTSOLResAttributes *winattrp;

Parameters
display Specifies a pointer to the Display structure; returned from XOpenDisplay().
object Specifies the ID of a window system object. Possible window system objects are windows and pixmaps.
type Specifies what type of resource is being accessed. Possible values are IsWindow and IsPixmap.
winattrp Client must provide a pointer to XTSOLResAttributes.

Description
The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. The XTSOLgetResAttributes() function retrieves all label attributes that are associated with a window or a pixmap in a single call. The attributes include UID, sensitivity label, and workstation owner.

Return Values
None.

Errors
BadAccess Lack of privilege.
BadWindow Not a valid window.
BadPixmap Not a valid pixmap.
BadValue Not a valid type.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also
libxtsol(3LIB), XTSOLgetClientAttributes(3XTSOL), XTSOLgetPropAttributes(3XTSOL), attributes(5)

“Obtaining Window Attributes” in Trusted Extensions Developer’s Guide
**Notes**  The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
XTSOLgetResLabel(3XTSOL)

Name  
XTSOLgetResLabel – get the label associated with a window, a pixmap, or a colormap

Synopsis  
cc [flag... file... -lX11 -lXtsol [library...]
#include <X11/extensions/Xtsol.h>

Status XTSOLgetResLabel(display, object, type, sl);

Display *display;
XID object;
ResourceType type;
m_label_t *sl;

Parameters  
display  Specifies a pointer to the Display structure; returned from XOpenDisplay().

object  Specifies the ID of a window system object whose label you want to get. Possible window system objects are windows, pixmaps, and colormaps.

type  Specifies what type of resource is being accessed. Possible values are IsWindow, IsPixmap or IsColormap.

sl  Returns a sensitivity label which is the current label of the specified object.

Description  
The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. The XTSOLgetResLabel() function retrieves the label that is associated with a window or a pixmap or a colormap.

Return Values  
None.

Errors  
BadAccess  Lack of privilege.

BadPixmap  Not a valid pixmap.

BadValue  Not a valid type.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also  
libXtsol(3LIB), XTSOLgetClientAttributes(3XTSOL), XTSOLsetResLabel(3XTSOL), attributes(5)

“Obtaining a Window Label” in Trusted Extensions Developer’s Guide

Notes  
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
Name  XTSOLgetResUID – get the UID associated with a window, a pixmap

Synopsis  cc [flag...] file... -LX11 -Lxtsol [library...]
          #include <X11/extensions/Xtsol.h>

          Status XTSOLgetResUID(display, object, type, uidp);

          Display *display;
          XID object;
          ResourceType type;
          uid_t *uidp;

Parameters  display  Specifies a pointer to the Display structure; returned from XOpenDisplay().

             object  Specifies the ID of a window system object whose UID you want to get. Possible window system objects are windows or pixmaps.

             type  Specifies what type of resource is being accessed. Possible values are IsWindow and IsPixmap.

             uidp  Returns a UID which is the current UID of the specified object. Client must provide a pointer to uid_t.

Description  The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. The XTSOLgetResUID() function retrieves the ownership of a window system object. This allows a client to get the ownership of an object that the client did not create.

Return Values  None.

Errors  BadAccess  Lack of privilege.

          BadWindow  Not a valid window.

          BadPixmap  Not a valid pixmap.

          BadValue  Not a valid type.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also  libXtso(3LIB), XTSOLgetClientAttributes(3XTSOL),
          XTSOLgetResAttributes(3XTSOL), XTSOLgetResLabel(3XTSOL), attributes(5)

"Obtaining the Window User ID" in Trusted Extensions Developer's Guide
Notes  The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
XTSOLgetSSHeight(3XTSOL)

**Name**
XTSOLgetSSHeight – get the height of screen stripe

**Synopsis**
```c
cc [flag...] file... -lX11 -lXtsol [library...]
#include <X11/extensions/Xtsol.h>

Status XTSOLgetSSHeight(display, screen_num, newheight);
```

- `display` Specifies a pointer to the Display structure; returned from XOpenDisplay().
- `screen_num` Specifies the screen number.
- `newheight` Specifies the storage area where the height of the stripe in pixels is returned.

**Description**
The XTSOLgetSSHeight() function gets the height of trusted screen stripe at the bottom of the screen. Currently the screen stripe is only present on the default screen. Client must have the Trusted Path process attribute.

**Return Values**
None.

**Errors**
- **BadAccess** Lack of privilege.
- **BadValue** Not a valid `screen_num` or `newheight`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**See Also**
libXtsol(3LIB), XTSOLsetSSHeight(3XTSOL), attributes(5)

“Accessing and Setting the Screen Stripe Height” in Trusted Extensions Developer’s Guide

**Notes**
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
Name  XTSOLgetWorkstationOwner – get the ownership of the workstation

Synopsis  cc [flag...] file... -lX11 -lXtsol [library...]
        #include <X11/extensions/Xtsol.h>

        Status XTSOLgetWorkstationOwner(display, uidp);

        Display *display;
        uid_t *uidp;

Parameters  display     Specifies a pointer to the Display structure; returned from XOpenDisplay().

            uidp        Returns a UID which is the current UID of the specified display workstation
                        server. Client must provide a pointer to uid_t.

Description  The XTSOLgetWorkstationOwner() function retrieves the ownership of the workstation.

Return Values  None.

Errors  BadAccess   Lack of privilege.

Attributes  See attributes(5) for descriptions of the following attributes:

+---------------------------------+-------------------+
| ATTRIBUTE TYPE                  | ATTRIBUTE VALUE   |
+---------------------------------+-------------------+
| Interface Stability             | Committed         |
| MT-Level                        | Unsafe            |
+---------------------------------+-------------------+

See Also  libXtsol(3LIB), XTSOLsetWorkstationOwner(3XTSOL), attributes(5)

“Obtaining the X Window Server Workstation Owner ID” in Trusted Extensions Developer’s Guide

Notes  The functionality described on this manual page is available only if the system is configured
        with Trusted Extensions.
The XTSOLIsWindowTrusted() function tests if a window is created by a trusted client. The window created by a trusted client has a special bit turned on. The client does not require any privilege to perform this operation.

**Parameters**

- **display** Specifies a pointer to the Display structure; returned from XOpenDisplay().
- **window** Specifies the ID of the window to be tested.

**Return Values**

- True If the window is created by a trusted client.
- BadWindow Not a valid window.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**See Also**

libXtsol(3LIB), attributes(5)

**Notes**

The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
**XTSOLMakeTPWindow** – make this window a Trusted Path window

**Synopsis**

```c
cc [flag...] file... -lX11 -lXtsol [library...]
#include <X11/extensions/Xtsol.h>

Status XTSOLMakeTPWindow(display, w);
Display *display;
Window w;
```

**Parameters**
- `display` specifies a pointer to the `Display` structure; returned from `XOpenDisplay()`.
- `w` specifies the ID of a window.

**Description**
The `XTSOLMakeTPWindow()` function makes a window a trusted path window. Trusted Path windows always remain on top of other windows. The client must have the Trusted Path process attribute set.

**Return Values**
None.

**Errors**
- `BadAccess` Lack of privilege.
- `BadWindow` Not a valid window.
- `BadValue` Not a valid type.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**See Also**
- `libXtsol(3LIB)`, `attributes(5)`

**Notes**
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
XTSOLsetPolyInstInfo – set polyinstantiation information

Synopsis

cc [flag...] file... -lX11 -lXtsol [library...]
#include <X11/extensions/Xtsol.h>

Status XTSOLsetPolyInstInfo(display, sl, uidp, enabled);

Display *display;
m_label_t sl;
uid_t *uidp;
int enabled;

Parameters
display Specifications a pointer to the Display structure; returned from XOpenDisplay().
sl Specifies the sensitivity label.
uidp Specifies the pointer to UID.
enabled Specifies whether client can set the property information retrieved.

Description

The XTSOLsetPolyInstInfo() function sets the polyinstantiated information to get property resources. By default, when a client requests property data for a polyinstantiated property, the data returned corresponds to the SL and UID of the requesting client. To get the property data associated with a property with specific sl and uid, a client can use this call to set the SL and UID with enabled flag to TRUE. The client should also restore the enabled flag to FALSE after retrieving the property value. Client must have the PRIV_WIN_MAC_WRITE and PRIV_WIN_DAC_WRITE privileges.

Return Values

None.

Errors

BadAccess Lack of privilege.
BadValue Not a valid display or sl.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also

libXtsol(3LIB), attributes(5)

“Setting Window Polyinstantiation Information” in Trusted Extensions Developer’s Guide

Notes

The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
XTSOLsetPropLabel(3XTSOL)

Name  XTSOLsetPropLabel – set the label associated with a property hanging on a window

Synopsis  cc [flag...] file... -lX11 -lXtsol [library...]
          #include <X11/extensions/Xtsol.h>

          Status XTSOLsetPropLabel(*display, window, property, *sl);

          Display *display;
          Window window;
          Atom property;
          m_label_t *sl;

Parameters  display  Specifies a pointer to the Display structure; returned from XOpenDisplay().

            window  Specifies the ID of the window whose property’s label you want to change.

            property  Specifies the property atom.

            sl  Specifies a pointer to a sensitivity label.

Description  The XTSOLsetPropLabel() function changes the sensitivity label that is associated with a
             property hanging on a window. The client must have the PRIV_WIN_DAC_WRITE,
             PRIV_WIN_MAC_WRITE, and PRIV_WIN_UPGRADE_SL privileges.

Return Values  None.

Errors  BadAccess  Lack of privilege.

            BadWindow  Not a valid window.

            BadAtom  Not a valid atom.

            BadValue  Not a valid sl.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also  libXtsol(3LIB), XTSOLgetPropAttributes(3XTSOL), XTSOLgetPropLabel(3XTSOL),
          attributes(5)

Notes  The functionality described on this manual page is available only if the system is configured
        with Trusted Extensions.
Name  XTSOLsetPropUID – set the UID associated with a property hanging on a window

Synopsis  cc [flag...] file... -lx11 -lxtsol [library...]
#include <X11/extensions/Xtsol.h>

Status XTSOLsetPropUID(display, window, property, uidp);

Display *display;
Window window;
Atom property;
uid_t *uidp;

Parameters  
display Specifies a pointer to the Display structure; returned from XOpenDisplay().

window Specifies the ID of the window whose property’s UID you want to change.

property Specifies the property atom.

uidp Specifies a pointer to a uid_t that contains a UID.

Description  The XTSOLsetPropUID() function changes the ownership of a window’s property. This allows another client to modify a property of a window that it did not create. The client must have the PRIV_WIN_DAC_WRITE and PRIV_WIN_MAC_WRITE privileges.

Return Values  None.

Errors  
BadAccess Lack of privilege.
BadWindow Not a valid window.
BadAtom Not a valid atom.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also  libXtsol(3LIB), XTSOLgetPropAttributes(3XTSOL), XTSOLgetPropUID(3XTSOL), attributes(5)

Notes  The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
XTSOLsetResLabel – set the label associated with a window or a pixmap

Synopsis  cc [flag...] file... -lX11 -lXtsol [library...]
          #include <X11/extensions/Xtsol.h>

          Status XTSOLsetResLabel(display, object, type, sl);
          Display *display;
          XID object;
          ResourceType type;
          m_label_t *sl;

Parameters
display Specifies a pointer to the Display structure; returned from XOpenDisplay().
object Specifies the ID of a window system object whose label you want to change.
Possible window system objects are windows and pixmaps.
type Specifies what type of resource is being accessed. Possible values are IsWindow and IsPixmap.
sl Specifies a pointer to a sensitivity label.

Description The client must have the PRIV_WIN_DAC_WRITE, PRIV_WIN_MAC_WRITE,
PRIV_WIN_UPGRADE_SL, and PRIV_WIN_DOWNGRADE_SL privileges. The XTSOLsetResLabel() function changes the label that is associated with a window or a pixmap.

Return Values None.

Errors BadAccess Lack of privilege.
BadPixmap Not a valid pixmap.
BadValue Not a valid type or sl.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also  libXtsol(3LIB), XTSOLgetResAttributes(3XTSOL), XTSOLgetResLabel(3XTSOL), attributes(5)

“Setting a Window Label” in Trusted Extensions Developer's Guide

Notes The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
## XTSOLsetResUID(3XTSOL)

### Name
XTSOLsetResUID – set the UID associated with a window, a pixmap, or a colormap

### Synopsis
```c
cc [flag...] file... -lx11 -lxtsol [library...]
#include <X11/extensions/Xtsol.h>

Status XTSOLsetResUID(display, object, type, uidp);

Display *display;
XID object;
ResourceType type;
uid_t *uidp;
```

### Parameters
- **display** Specifies a pointer to the `Display` structure; returned from `XOpenDisplay()`.
- **object** Specifies the ID of a window system object whose UID you want to change. Possible window system objects are windows and pixmaps.
- **type** Specifies what type of resource is being accessed. Possible values are: `IsWindow` and `IsPixmap`.
- **uidp** Specifies a pointer to a `uid_t` structure that contains a UID.

### Description
The client must have the `PRIV_WIN_DAC_WRITE` and `PRIV_WIN_MAC_WRITE` privileges. The `XTSOLsetResUID()` function changes the ownership of a window system object. This allows a client to create an object and then change its ownership. The new owner can then make modifications on this object as this object being created by itself.

### Return Values
None.

### Errors
- **BadAccess** Lack of privilege.
- **BadWindow** Not a valid window.
- **BadPixmap** Not a valid pixmap.
- **BadValue** Not a valid type.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Unsafe</td>
</tr>
</tbody>
</table>

### See Also
`libXtsol(3LIB), XTSOLgetResUID(3XTSOL), attributes(5)`

### Notes
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
XTSOLsetSessionHI(3XTSOL)

Name
XTSOLsetSessionHI – set the session high sensitivity label to the window server

Synopsis
cc [flag...] file... -lx11 -lxtsol [library...]
#include <x11/extensions/xtsol.h>

Status XTSOLsetSessionHI(display, sl);

Display *display;
m_label_t *sl;

Parameters
display Specifies a pointer to the Display structure; returned from XOpenDisplay().
sl Specifies a pointer to a sensitivity label to be used as the session high label.

Description
The XTSOLsetSessionHI() function sets the session high sensitivity label. After the session
high label has been set by a Trusted Extensions window system TCB component, logintool,
X server will reject connection request from clients running at higher sensitivity labels than
the session high label. The client must have the PRIV_MIN_CONFIG privilege.

Return Values
None.

Errors
BadAccess Lack of privilege.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also
libXtsol(3LIB), XTSOLsetSessionLO(3XTSOL), attributes(5)

“Setting the X Window Server Clearance and Minimum Label” in Trusted Extensions
Developer’s Guide

Notes
The functionality described on this manual page is available only if the system is configured
with Trusted Extensions.
Name
XTSOLsetSessionLO – set the session low sensitivity label to the window server

Synopsis
cc [flag...] file... -lX11 -lxtsol [library...]
#include <X11/extensions/Xtsol.h>

Status XTSOLsetSessionLO(display, sl);

Display *display;
mlabel_t *sl;

Parameters
display Specifies a pointer to the Display structure; returned from XOpenDisplay().
sl Specifies a pointer to a sensitivity label to be used as the session low label.

Description
The XTSOLsetSessionLO() function sets the session low sensitivity label. After the session low label has been set by a Trusted Extensions window system TCB component, logintool, X server will reject a connection request from a client running at a lower sensitivity label than the session low label. The client must have the PRIV_WIN_CONFIG privilege.

Return Values
None.

Errors
BadAccess Lack of privilege.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also
libXtsol(3LIB), XTSOLsetSessionHI(3XTSOL), attributes(5)

"Setting the X Window Server Clearance and Minimum Label" in Trusted Extensions Developer’s Guide

Notes
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
Name  XTSOLsetSSHeight – set the height of screen stripe

Synopsis  cc [flag...] file... -lX11 -lXtsol [library...]
          #include <X11/extensions/Xtsol.h>

          Status XTSOLsetSSHeight(display, screen_num, newheight);

          Display *display;
          int screen_num;
          int newheight;

Parameters  display       Specifies a pointer to the Display structure; returned from XOpenDisplay.

            screen_num    Specifies the screen number.

            newheight     Specifies the height of the stripe in pixels.

Description  The XTSOLsetSSHeight() function sets the height of the trusted screen stripe at the bottom of
              the screen. Currently the screen stripe is present only on the default screen. The client must
              have the Trusted Path process attribute.

Return Values  None.

Errors  BadAccess    Lack of privilege.

           BadValue     Not a valid screen_num or newheight.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

See Also  libXtsol(3LIB), XTSOLgetSSHeight(3XTSOL), attributes(5)

"Accessing and Setting the Screen Stripe Height" in Trusted Extensions Developer's Guide

Notes  The functionality described on this manual page is available only if the system is configured
        with Trusted Extensions.
**Name**  
XTSOLsetWorkstationOwner - set the ownership of the workstation

**Synopsis**  
cc [flag...] file... -lX11 -lXtsol [library...]  
#include <X11/extensions/Xtsol.h>  

Status XTSOLsetWorkstationOwner(display, uidp);

Display *display;  
uid_t *uidp;  
XTSOLClientAttributes *clientattrp;

**Parameters**  
display  
Specifies a pointer to the Display structure; returned from XOpenDisplay().

uidp  
Specifies a pointer to a uid_t structure that contains a UID.

**Description**  
The XTSOLsetWorkstationOwner() function is used by the Solaris Trusted Extensions logintool to assign a user ID to be identified as the owner of the workstation server. The client running under this user ID can set the server’s device objects, such as keyboard mapping, mouse mapping, and modifier mapping. The client must have the Trusted Path process attribute.

**Return Values**  
None.

**Errors**  
BadAccess  
Lack of privilege.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**See Also**  
libXtsol(3LIB), XTSOLgetWorkstationOwner(3XTSOL), attributes(5)

“Accessing and Setting a Workstation Owner ID” in Trusted Extensions Developer’s Guide

**Notes**  
The functionality described on this manual page is available only if the system is configured with Trusted Extensions.
zs_open\(3\)ZONESTAT

**Name**  
zs_open, zs_close – open and close the zones statistics facility

**Synopsis**  
cc [ flag ... ] file... -lzonestat [ library ... ]  
#include <zonestat.h>

zs_ctl_t zs_open();  
void zs_close(zs_ctl_t zctl);

**Description**  
The `zs_open()` function connects to the zones statistic facility and returns a `zonestat` control object, which can then be used to read zone utilization information.

All reads using a `zonestat` control object are relative to the point in time in which the object was returned by `zs_open()`. For example, CPU usage will be reported as usage since the `zs_open()` call was made.

The `zs_close()` function closes the connection to the zones statistics facility and frees all associated resources.

**Return Values**  
On success, `zs_open()` returns a pointer to a `zonestat` control object. On failure, `zs_open()` returns `NULL` and sets `errno` to indicate the error.

**Errors**  
The `zs_open()` function will fail if:

- **EAGAIN** There are insufficient resources available.
- **EMFILE** Opening would exceed the maximum number of file descriptors allowed for the current process.
- **ENOMEM** There is insufficient memory available.
- **EPERM** The caller does not have privilege `proc_info`.
- **ESRCH** Unable to connect to the zones monitoring service. See Notes below.

**Attributes**  
See attributes\((5)\) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>See below.</td>
</tr>
</tbody>
</table>

The `zs_open()` and `zs_close()` functions are MT-safe, with the exception that only one thread may actively use a `zs_ctl_t` object at any time. Synchronization is left to the application.
See Also  zonestat(1), pooladm(1M), psrset(1M), rcapadm(1M), swap(1M), zoneadm(1M),
zonestatd(1M), libpool(3LIB), libzonestat(3LIB), zs_pset(3ZONESTAT),
zs_property(3ZONESTAT), zs_pset_zone(3ZONESTAT), zs_resource(3ZONESTAT),
zs_usage(3ZONESTAT), zs_zone(3ZONESTAT), attributes(5), resource_controls(5)

Notes  The service svc:/system/zones-monitoring:default must be enabled in the global zone in
order for zs_open() to succeed. This requirement exists for use of libzonestat in both the
global zone and non-global zones.
zs_property(3ZONESTAT)

**Name**  
zs_property, zs_property_type, zs_property_string, zs_property_double, zs_property_uint64, zs_property_int64, zs_property_uint, zs_property_int – libzonestat property value accessor methods

**Synopsis**  
cc [ flag ... ] file ... -lzonestat [ library ... ]
#include <zonestat.h>

data_type_t zs_property_type(zs_property_t property);
char *zs_property_string(zs_property_t property);
double zs_property_double(zs_property_t property);
uint64_t zs_property_uint64(zs_property_t property);
int64_t zs_property_int64(zs_property_t property);
uint_t zs_property_uint(zs_property_t property);
int zs_property_int(zs_property_t property);

**Description**  
These functions retrieve the values from zs_property_t objects.

The zs_property_type() function returns a value from the data_type_t enumeration defined in <sys/nvpair.h>, which is one of:

<table>
<thead>
<tr>
<th>Value</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_TYPE_STRING</td>
<td>char *</td>
</tr>
<tr>
<td>DATA_TYPE_UINT64</td>
<td>uint64_t</td>
</tr>
<tr>
<td>DATA_TYPE_INT64</td>
<td>int64_t</td>
</tr>
<tr>
<td>DATA_TYPE_UINT32</td>
<td>uint_t</td>
</tr>
<tr>
<td>DATA_TYPE_INT32</td>
<td>int</td>
</tr>
<tr>
<td>DATA_TYPE_DOUBLE</td>
<td>double</td>
</tr>
</tbody>
</table>

The zs_property_string() function returns a char * pointing to a string. Callers must copy this string in order to retain it after freeing the zs_usage_t object from which this property was obtained.

The zs_property_uint64(), zs_property_int64(), zs_property_uint(), zs_property_int(), and zs_property_double() functions return the value contained in the property object.

**Return Values**  
See Description.
If a `zs_property_*` function is called with a property of the wrong type, the function will abort with `abort(3C)`.

**Examples** See `zs_pset_zone(3ZONESTAT)`.

**Attributes** See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**See Also** `zonestat(1), pooladm(1M), psrset(1M), rcapadm(1M), swap(1M), zoneadm(1M),
zonestatd(1M), abort(3C), libpool(3LIB), libzonestat(3LIB), zs_pset(3ZONESTAT),
zs_open(3ZONESTAT), zs_pset_zone(3ZONESTAT), zs_resource(3ZONESTAT),
zs_usage(3ZONESTAT), zs_zone(3ZONESTAT), attributes(5), resource_controls(5)`
zs_pset_list, zs_pset_property, zs_pset_total_time, zs_pset_total_cpus, zs_pset_used_time, zs_pset_used_cpus, zs_pset_used_pct – libzonestat pset accessor methods

Synopsis

cc [ flag ... ] file... -lzonestat [ library... ]
#include <zonestat.h>

int zs_pset_list(zs_usage_t usage, zs_pset_t *psetlist, int num);
zs_pset_t zs_pset_walk(zs_usage_t usage, zs_pset_t pset);
zs_property_t zs_pset_property(zs_pset_t pset, zs_pset_property_t prop);
void zs_pset_total_time(zs_pset_t pset, timestruc_t *ts);
uint64_t zs_pset_total_cpus(zs_pset_t pset);
void zs_pset_used_time(zs_pset_t pset, zs_user_t user, timestruc_t *ts);
uint64_t zs_pset_used_cpus(zs_pset_t pset, zs_user_t user);
uint_t zs_pset_used_pct(zs_pset_t pset, zs_user_t user);

Description

These functions are used to access the processor sets (psets) in the usage object. These psets are the psets which existed at the time when the zs_usage_read was called, and returned the usage object.

By default, the system has a single processor set which contains all cpus. Processor sets can be created by resource pools (pooladm(1M)), the psset(1M) command, and by the zonecfg(1M) dedicated cpu resource.

The zs_pset_list function returns the number of zs_pset_t objects contained within usage. If psetlist is non-NULL, the psetlist array will be filled with up to num zs_pset_t objects. The psetlist array must be first allocated by the caller. The first pset returned in the array will always be the default pset. The remaining pset will be in alphanumeric ordered by pset name.

The zs_pset_walk function walks the psets contained in usage. If pset is NULL, the first pset is returned. The first pset is always the default pset. Otherwise, the psets are returned in alphanumeric order. NULL will be returned if there are no more psets.

The zs_pset_property function will return the prop property of a pset. See libzonestat(3lib) for a description of the ZS_PSET_PROP_* property codes.

The zs_pset_total_time function sets ts to the total cpu time that has been available in pset. For example, if a pset has 2 online cpus, and zs_open as called 30 seconds before zs_usage_read, then the total time available in the pset is 60 seconds. ts must be allocated by the caller.

The zs_pset_total_cpus function returns the number of online cpus in pset * 100. For instance, if the number of online cpus is 4, the value returned will be 400.

The zs_pset_used_time function sets ts to the total cpu time that has been used in pset, starting at zero from the point when zs_open was first called.
The `zs_pset_used_cpus` function returns the quantity of CPUs used by user. The value returned is the number of CPUs used * 100. For instance, if user `ZS_USER_ZONES` (which represents total pset usage by all zones) has used 2.5 CPUs worth of CPU time, then the value returned is 250.

The `uint_t zs_pset_used_pct` function returns the percentage of CPU time used by user. The value returned is the percentage * 100. For example, if user `ZS_USER_ALL` (representing total CPU usage) is 50%, then the value returned will be 5000.

All `ZS_USER_*` user codes are described in the `libzonestat(3lib)` manual page.

**Return Values**

See Description

**Errors**

If a `zs_pset_*()` function is called with an invalid user code, the function will abort with `abort(3C)`.

**Examples**

**EXAMPLE 1** Retrieve information about all psets in a usage object.

The following example traverses all psets in a usage object, retrieving information about each pset.

```c
#include <zonestat.h>
...
extern zs_usage_t usage;  /* assume returned by zs_usage_read() */
...
zs_pset_t pset;
zs_property_t prop;
char * psetname;
uint64_t online;
uint64_t used;

for (pset = zs_pset_first(usage); pset != NULL; 
pset = zs_pset_next(usage, pset)) {
    /* Get psetname */
    prop = zs_pset_property(pset, ZS_PSET_PROP_NAME);
    psetname = strdup(zs_property_string(prop));

    /* Get number of online CPUs, and quantity of CPU used, both in number of CPUs * 100.
    */
    online = zs_pset_total_cpus(pset);
    used = zs_pset_used_cpus(pset, ZS_USER_ALL);
}
```

**Attributes**

See `attributes(5)` for descriptions of the following attributes:
See Also  
zonestat(1), pooladm(1M), psrset(1M), rcapadm(1M), swap(1M), zoneadm(1M),  
zonestatd(1M), abort(3C), libpool(3LIB), libzonestat(3LIB), zs_open(3ZONESTAT),  
zs_property(3ZONESTAT), zs_pset_zone(3ZONESTAT), zs_resource(3ZONESTAT),  
zs_usage(3ZONESTAT), zs_zone(3ZONESTAT), attributes(5), resource_controls(5)
These functions are used to access the per-zone utilization information for a given pset. Each pset have zero or more zones bound to it.

Typically, a zone will be bound to a single pset, but it is possible for the global zone to be bound to multiple psets, as well as non-global zones if `prset(1M)` pssets are used.

If a zone is bound to multiple pssets, its CPUs shares will be applied to each pset. For instance, if a zone has 10 shares, and has processes in both pset A and pset B, then the zone will have 10 shares in both pset A and pset B. The relative value of those shares will depend on the other zones running in each pset, and how many shares the other zones have.

If a zone is bound to multiple pssets, then its CPU cap is applied across all pssets. The zone may use up to its cap in CPU time. Some of this time could be spent in pset A, and the rest in pset B. Each zone does not get its full CPU cap per pset.

The `zs_pset_zone_list()` function returns the number of `zs_pset_zone_t` objects contained within `pset`. If `pzlist` is non-NULL, the `pzlist` array will be filled with up to `num` `zs_pset_zone_t` objects. The `pzlist` array must be first allocated by the caller. The `zs_pset_zone_t` objects will be returned in alphanumeric order by zone name.

The `zs_pset_zone_walk()` function is used to walk the list of zones using a `pset`. Zones are walked in alphanumeric order. If `pz` is NULL, the first zone is returned, otherwise the zone after `pz` is returned. NULL is returned if there are no more zones using the `pset`.

The `zs_pset_zone_get_zone()` function returns the `zs_zone_t` object representing the zone for `pz`. 

```c
int zs_pset_zone_list(zs_pset_t pset, zs_pset_zone_t *pzlist, int num);
zs_pset_zone_t zs_pset_zone_walk(zs_pset_t pset, zs_pset_zone_t pz);
zs_zone_t zs_pset_zone_get_zone(zs_pset_zone_t pset);
zs_pset_zone_t zs_pset_zone_get_pset(zs_pset_zone_t pz);
zs_property_t zs_pset_zone_property(zs_pset_zone_t pz, zs_pz_property_t prop);
void zs_pset_zone_used_time(zs_pset_zone_t pz, timespec_t *ts);
uint64_t zs_pset_zone_used_cpus(zs_pset_zone_t pz);
uint_t zs_pset_zone_used_pct(zs_pset_zone_t pz, zs_pz_pct_t pct);
```
The `zs_pset_zone_get_pset()` function returns the `zs_pset_t` object representing the pset for `pz`.

The `zs_pset_zone_property()` function will return the `prop` property of a `pz`. See `libzonestat(3LIB)` for a description of the `ZS_PZ_PROP_*` property codes.

The `zs_pset_zone_used_time()` function sets `ts` to the total CPU time that has been used in the pset by the zone, starting at zero from the point when `zs_open(3ZONESTAT)` was first called.

The `zs_pset_zone_used_cpus()` function sets `ts` to the total number of CPUs that has been used in the pset by the zone, starting at zero from the point when `zs_open()` was first called. The value returned is the number of CPUs used \* 100. For example, if the zone used 1.5 CPUs in the pset, the returned value will be 1500.

The `zs_pset_zone_used_pct()` function returns the percentage of `pct` used by the zone in the pset. See the description of the `ZS_PZ_PCT_*` codes on the `libzonestat(3LIB)` manual page.

Return Values

See Description.

Errors

If a `zs_pset_zone_*()` function is called with an invalid property or `pct` code, the function will abort with `abort(3C)`.

Examples

**EXAMPLE 1**  Walk the list of zones and retrieve CPU data.

The following example walks the list of zones using the default pset, retrieving both the CPU shares and CPU cap, as well as the percent of each used.

```c
#include <zonestat.h>
#include <strings.h>
...
extern zs_usage_t usage; /* assume returned by zs_usage_read() */
...
zs_pset_t pset;
zs_pset_zone_t pz;
zs_property_t prop;
char *psetname;
char *zonename;
uint64_t shares;
uint64_t cap;
uint_t pct_shares;
uint_t pct_cap;

/* Get default pset and name, which is always the first pset */
pset = zs_pset_first(usage);
zs_pset_property(pset, ZS_PROP_PSET_NAME prop);
psetname = strdup(zs_property_string(prop));

for (pz = zs_pset_zone_first(pset); pz != NULL;)
```
EXAMPLE 1 Walk the list of zones and retrieve CPU data.  

```c
pz = zs_pset_zone_next(pset, pz) {
    /* Get name of zone */
    zone = zs_pset_zone_get_zone(pz);
    prop = zs_zone_property(pset, ZS_PROP_ZONE_NAME);
    zonename = strdup(zs_property_string(prop));

    /* get shares and cap */
    prop = zs_pset_zone_property(pz, ZS_PZ_PROP_CPU_SHARES);
    shares = zs_property_uint64(prop);
    prop = zs_pset_zone_property(pz, ZS_PZ_PROP_CPU_CAP);
    cap = zs_property_uint64(prop);

    /* get percent used of shares and cap */
    pct_shares = zs_pset_zone_used_pct(pz, ZS_PZ_PCT_CPU_SHARE);
    pct_cap = zs_pset_used_cpus(pz, ZS_PZ_PCT_CPU_CAP);
}
```

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  
zonectat(1), pooladm(1M), psrset(1M), rcapadm(1M), swap(1M), zoneadm(1M),  
zonectatd(1M), abort(3C), libpool(3LIB), libzonectat(3LIB), zs_open(3ZONESTAT),  
zs_property(3ZONESTAT), zs_pset(3ZONESTAT), zs_resource(3ZONESTAT),  
zs_usage(3ZONESTAT), zs_zone(3ZONESTAT), attributes(5), resource_controls(5)
Name
zs_resource, zs_resource_property, zs_resource_type, zs_resource_total_uint64,
zs_resource_total_time, zs_resource_used_uint64, zs_resource_used_time,
zs_resource_used_pct, zs_resource_used_zone_uint64, zs_resource_used_zone_time,
zs_resource_used_zone_pct

Synopsis
cc [ flag ... ] file... -lzonestat [ library ... ]
#include <zonestat.h>

zs_property_t zs_resource_property(zs_usage_t usage,
    zs_resource_property_t prop);
zs_resource_type_t zs_resource_type(zs_resource_t resource);
uint64_t zs_resource_total_uint64(zs_usage_t usage,
    zs_resource_t resource);
void zs_resource_total_time(zs_usage_t usage,
    zs_resource_t resource, timestruc_t *ts);
uint64_t zs_resource_used_uint64(zs_usage_t usage,
    zs_resource_t resource, zs_user_t user);
void zs_resource_used_time(zs_usage_t usage, zs_resource_t resource,
    zs_user_t user, timestruc_t *ts);
uint_t zs_resource_used_pct(zs_usage_t usage, zs_resource_t resource,
    zs_user_t user);
uint64_t zs_resource_used_zone_uint64(zs_zone_t zone,
    zs_resource_t resource);
void zs_resource_used_zone_time(zs_zone_t zone, zs_resource_t resource,
    timestruc_t *ts);
uint_t zs_resource_used_zone_pct(zs_zone_t zone, zs_resource_t resource);

Description
These functions are used to access properties and utilization information of the system
resources. Also provided by the zs_resource_used_zone_*( ) functions is per-zone
utilization information of each resource.

See libzonestat(3LIB) for the ZS_RESOURCE_ * resource codes and the ZS_USER_ * user codes.

The zs_resource_property( ) function returns resource property prop. See
libzonestat(3LIB) for a description of the ZSRESOURCE_PROP_ * property codes.

The zs_resource_type( ) function returns the data type of the resource. The following types
can be returned:

ZS_RESOURCE_TYPE_TIME
  The resource and its usage can be retrieved in terms of time. This limit can be passed to
  zs_resource_total_time() and zs_resource_used_time(), as well as all other
  zs_resource_*( ) functions.
ZS_RESOURCE_TYPE_COUNT
   The resource reflects a quantity of discrete objects. For instance, a limit on the number of processes.

ZS_RESOURCE_TYPE_BYTES
   The resource reflects a quantity of bytes.

The `zs_resource_total_uint64()` function returns the total resource available of type `resource`.

The `zs_resource_total_time()` function sets `ts` to the total time available for the given resource. This is supported only for the ZSRESOURCE_CPU resource. The total CPU time represents the total CPU time available since `zs_open(3ZONESTAT)` was called.

The `zs_resource_used_uint64()` function returns the total resource used for the given resource by the given user.

The `zs_resource_used_time()` function sets `ts` to the total time used for `resource` by `user`. This is supported only for the ZSRESOURCE_CPU resource. The used CPU time represents the CPU time used since `zs_open()` was called.

The `zs_resource_used_pct()` function returns the percentage of resource used by `user`.

The `zs_resource_used_zone_uint64()` function returns the quantity of `resource` by `zone`.

The `zs_resource_used_zone_time()` function returns the quantity of `resource` time used by `zone`. This usage value is increasing from when `zs_open()` as first called. This function supports only the ZSRESOURCE_CPU resource.

The `zs_resource_used_zone_pct()` function returns the percent of `resource` used by `zone`.

**Return Values**

See Description.

**Errors**

If a `zs_resource_*()` function is called with an invalid resource or user code, the function will abort with `abort(3C)`.

**Examples**

**EXAMPLE 1** Retrieve physical memory.

The following example retrieves physical memory utilization both system-wide and for each zone.

```c
#include <zonestat.h>
...
extern zs_usage_t usage; /* assume returned by zs_usage_read() */
...
zs_zone_t zone;
uint64_t total_memory;
uint64_t used_memory;
uint64_t zone_used_memory;
```
EXAMPLE 1  Retrieve physical memory.  

(Continued)

total_memory = zs_resource_total_uint64(usage, ZS_RESOURCE_RAM);
used_memory = zs_resource_used_uint64(usage, ZS_RESOURCE_RAM, ZS_USER_ALL);

for (zone = zs_zone_first(usage); zone != NULL;
    zone = zs_zone_next(usage, zone)) {

    zone_used_memory = zs_resource_used_zone_uint64(zone, ZSRESOURCE_RAM);
}
#include <zonestat.h>

zs_usage_t zs_usage_read(zs_ctl_t zsctl);

zs_usage_t zs_usage_diff(zs_usage_t u1, zs_usage_t u2);

void zs_usage_free(zs_usage_t zsctl);

The `zs_usage_read()` function reads the system configuration and utilization with respect to zones, memory, and CPU.

Each `zs_usage_read()` will return a `zs_usage_t` object, which includes the following information:
- The current system resources and their utilization.
- The currently running zones. All properties reflect the current running state of each zone.
- The usage of each resource by each zone.
- The usage by each zone of its configured limits.
- The currently existing processor sets. All properties and usages reflect the current running state of each processor set.
- The usage of each processor set by each zone utilizing it.

Increasing utilization values are described in `libzonestat(3LIB)`. Increasing values continually increase, starting at zero from the point at which `zs_open(3ZONESTAT)` was first called.

For all other utilization values, utilization information will be the usage at the point in time at which `zs_usage_read()` is called.

For zones and processors sets that were booted or created after `zs_open()` was called, the increasing usage values will be usage since the most recent boot or creation before the call to `zs_usage_read()`.

The `zs_usage_diff()` function computes and returns the utilization differences between two `zs_usage_t` objects returned by `zs_usage_read()`. Both `u1` and `u2` must be read from the same `zsctl` object. `u2` must be read after `u1`.

The purpose of `zs_usage_diff()` is to simplify the comparison of `zs_usage_read()` calls made at a interval.

The returned `zs_usage_t` object will contain:
- The current system resources, as they exist in `u2`.
- The zones and processor sets that exist in `u2`.

---

Extended Library Functions, Volume 4 367
For increasing utilization values, the value of \((u_2 - u_1)\). If a specific value does not exist in \(u_1\) (such as the CPU utilization for a zone in \(u_2\) which does not exist in \(u_1\)), the value will be the value from \(u_2\).

For non-increasing utilization values, the value of \(u_2\).

The \texttt{zs\_usage\_free()} function frees the memory associated with a \texttt{zs\_usage\_t} object returned by \texttt{zs\_usage\_read()} or \texttt{zs\_usage\_diff()}.

**Return Values**

On success, \texttt{zs\_usage\_free()} and \texttt{zs\_usage\_diff()} return a pointer to a zonestat usage object. On failure, \texttt{zs\_open()} returns \texttt{NULL}.

**Errors**

The \texttt{zs\_usage\_diff()} function will fail if:

- \texttt{EAGAIN} Insufficient resources are available.
- \texttt{ENOMEM} Insufficient memory is available.

The \texttt{zs\_usage\_read()} function will fail if:

- \texttt{EAGAIN} Insufficient resources are available.
- \texttt{EINTR} A signal was caught.
- \texttt{ENOMEM} Insufficient memory is available.
- \texttt{ESRCH} Unable to connect to the zones monitoring service. See Notes.

**Examples**

**EXAMPLE 1**

Read zone CPU utilization.

The following example uses \texttt{zs\_usage\_read()} to read each zones CPU utilization at a regular interval.

```c
#include "zonestat.h"
...
extern int time_to_quit; /* hypothetical quit notification */
...
zs_ctl_t ctl;
zs_usage_t last;
zs_usage_t next;
zs_usage_t diff;
zs_zone_t zone;
timestruct_t time;
uint64_t cpus;

ctl = zs_usage_open(); /* Open zone statistics */
last = zs_usage_read(); /* Read initial usage */
for (;;) {
    sleep(10);
```
EXAMPLE 1  Read zone CPU utilization.  

```
next = zs_usage_read();
diff = zs_usage_diff(last, next);

/* Walk zones in usage data */
for (zone = zs_zone_first(diff); zone != NULL ;
    zone = zs_zone_next(diff, zone)) {

    /*
     * fetch cpu time used by zone over interval in terms of
     * seconds and nanoseconds
     */
    zs_resource_used_zone_time(zone, ZS_RESOURCE_CPU,
        &time);

    /*
     * fetch cpu time used by zone over interval in terms of
     * cpu used. 100 equals one cpu. For example, a
     * value of 250 means the zone used 2.5 cpus worth of
     * time between last and next.
     */
    zs_resource_used_zone_uint64(zone, ZS_RESOURCE_CPU,
        &cpus;

}  
zs_usage_free(diff);
zs_usage_free(last);  
last = next;
if (time_to_quit)
    break;
}
zs_usage_free(last)
```

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>See below.</td>
</tr>
</tbody>
</table>

The zs_usage_*() functions are MT-safe, with the exception that only one thread may actively use a zs_ctl_t* object at any time. Synchronization is left to the application.
See Also  zonestat(1), pooladm(1M), psrset(1M), rcapadm(1M), swap(1M), zoneadm(1M),
         zonestatd(1M), libpool(3LIB), libzonestat(3LIB), zs_open(3ZONESTAT),
         zs_pset(3ZONESTAT), zs_property(3ZONESTAT), zs_pset_zone(3ZONESTAT),
         zs_resource(3ZONESTAT), zs_zone(3ZONESTAT), attributes(5),
         resource_controls(5)

Notes  The service svc:/system/zones-monitoring:default must be enabled in the global zone in
       order for zs_usage_read() to succeed. This requirement exists for use of libzonestat(3LIB)
       in both the global zone and non-global zones.

       If the zones-monitoring service goes off line, ESRCH will be returned. At this point the zsctl
       object is no longer usable. Use zs_close(zsctl) and use zs_open(3ZONESTAT) to reconnect.
zs_zone(3ZONESTAT)

Name  
zs_zone, zs_zone_list, zs_zone_walk, zs_zone_property, zs_zone_limit_type,
zs_zone_limit_uint64, zs_zone_limit_used_uint64, zs_zone_limit_time,
zs_zone_limit_used_time, zs_zone_limit_used_pct  - libzonestat zone accessor methods

Synopsis  
cc [ flag ... ] file... -lzonestat [ library ... ]
#include <zonestat.h>

int zs_zone_list(zs_usage_t usage, zs_zone_t *zonelist, int num);
zs_zone_t zs_zone_walk(zs_usage_t usage, zs_zone_t *zone);
zs_property_t zs_zone_property(zs_zone_t *zone, 
   zs_zone_property_t *prop);
zs_limit_type_t zs_zone_limit_type(zs_limit_t limit);
uint64_t zs_zone_limit_uint64(zs_zone_t *zone, zs_limit_t limit);
uint64_t zs_zone_limit_used_uint64(zs_zone_t *zone, zs_limit_t limit);
void zs_zone_limit_time(zs_zone_t *zone, zs_limit_t *limit, 
   timestruc_t *ts);
void zs_zone_limit_used_time(zs_zone_t *zone, zs_limit_t *limit, 
   timestruc_t *ts);
uint_t zs_zone_limit_used_pct(zs_zone_t *zone, zs_limit_t limit);

Description  
These functions are used to access the zones in the usage object. These zones are the zones
which were running at the time when the zs_usage_read(3ZONESTAT) was called and
returned the usage object.

The zs_zone_list() function returns the number of zs_zone_t objects contained within
usage. If zonelist is non-null, the zonelist array will be filled with up to num zs_zone_t objects.
The zonelist array must be first allocated by the caller. The first zone returned in the array will
always be the global zone. The remaining zones will be in alphanumeric ordered by zone
name.

The zs_zone_walk() function walks the list of zones in usage. If zone is NULL, the first zone
will be returned, which is always the global zone. Otherwise, zones are returned in
alphanumeric order. If there are no more zones in usage, NULL is returned.

The zs_zone_property() function returns a property of zone based on property. See
libzonestat(3LIB) for a description of the ZS_ZONE_PROP_* property codes.

The zs_zone_limit_type() function returns the data type of the zone limit limit. See
libzonestat(3LIB) for the description of the ZS_LIMIT_* limit codes. The following types can
be returned:
ZS_LIMIT_TYPE_TIME
The limit and its usage can be fetched in terms of time. This limit can be passed to `zs_zone_limit_time()` and `zs_zone_limit_used_time()`, as well as all other `zs_zone_limit_*()` functions.

ZS_LIMIT_TYPE_COUNT
The limit reflects a quantity of discrete objects. For instance, a limit on the number of processes.

ZS_LIMIT_TYPE_BYTES
The limit reflects a quantity of bytes.

The `zs_zone_limit_uint64()` function returns the value of the configured `limit` on `zone`. If `limit` is not configured for `zone`, `ZS_LIMIT_NONE` is returned. For example, `zone` may not be configured with the `ZS_LIMIT_CPU` limit.

The `zs_zone_limit_used_uint64()` function returns the usage by `zone` of the resource capped by `limit`. If `limit` is not configured for `zone`, `ZS_LIMIT_NONE` is returned. For example, `zone` may not be configured with the `ZS_LIMIT_CPU` limit.

The `zs_zone_limit_time()` function sets `ts` to the quantity of time available to `zone` by the configured `limit`. `ts` must be allocated by the caller. For example, if `zone` has a `ZS_LIMIT_CPU` of 200, meaning 2 CPUs and the zone has been running for 10 seconds, then the quantity of time available to the zone would be 20 seconds (2 CPUs * 10 seconds). If `limit` is not configured for the zone, `ts` is set to zero seconds and nanoseconds. This function can be called only if `limit` is of type `ZS_LIMIT_TYPE_TIME`.

The `zs_zone_limit_used_time()` function sets `ts` to the quantity of time used by `zone` on the resource capped by `limit`. `ts` must be allocated by the caller. If `limit` is not configured for the zone, `ts` is set to zero seconds and nanoseconds. This function can be called only if `limit` is of type `ZS_LIMIT_TYPE_TIME`.

The `zs_zone_limit_used_pct()` function returns the percent of `limit` used by `zone`. The value returned is the percentage * 100. For instance, 50 percent is returned as 5000. If `limit` is not configured for `zone`, `ZS_PCT_NONE` is returned.

All the ZS_LIMIT_* limit codes are described in `libzonestat(3LIB)`.

Return Values
See Description.

Errors
If a `zs_zone_*()` function is called with an invalid limit code, the function will abort with `abort(3C)`.

Examples
EXAMPLE 1 Retrieve information about each zone in a usage object.
The following example traverses all zones in a usage object, retrieving information about each zone.

```
#include <zonestat.h>
...
```
EXAMPLE 1  Retrieve information about each zone in a usage object.  (Continued)

extern zs_usage_t usage;  /* assume returned by zs_usage_read() */
...
zs_zone_t zone;
zs_property_t prop;
char * zonename;
uint64_t maxprocs;
uint64_t usedprocs;

for (zone = zs_zone_first(usage); zone != NULL; 
    zone = zs_zone_next(usage, zone)) {

  /* Get zonename */
  prop = zs_zone_property(zone, ZS_ZONE_PROP_NAME);
  zonename = zs_property_string(prop));

  /* Get max and used processes */
  maxprocs = zs_zone_limit_uint64(zone, ZS_LIMIT_PROCESSES);
  usedprocs = zs_zone_limit_used_uint64(zone, ZS_LIMIT_PROCESSES);
}

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

See Also  zonestat(1), pooladm(1M), psrset(1M), rcapadm(1M), swap(1M), zoneadm(1M), zonestatd(1M), abort(3C), libpool(3LIB), libzonestat(3LIB), zs_open(3ZONESTAT), zs_pset(3ZONESTAT), zs_property(3ZONESTAT), zs_pset_zone(3ZONESTAT), zs_resource(3ZONESTAT), zs_usage(3ZONESTAT), zs_open(3ZONESTAT), attributes(5), resource_controls(5)